



C o m m u n i t y E x p e r i e n c e D i s t i l l e d

The Android Game Developer's Handbook

Discover an all in one handbook to developing immersive and cross-platform Android games

Avisekhar Roy

[PACKT] open source*
PUBLISHING community experience distilled

www.allitebooks.com

The Android Game Developer's Handbook

Discover an all in one handbook to developing
immersive and cross-platform Android games

Avisekhar Roy

[PACKT] open source 
PUBLISHING community experience distilled

BIRMINGHAM - MUMBAI

The Android Game Developer's Handbook

Copyright © 2016 Packt Publishing

retrieval
itten
dded in
critical articles or reviews.

accuracy
book is
r Packt
damages
caused or alleged to be caused directly or indirectly by this book.

ll of the
companies and products mentioned in this book by the appropriate use of capitals.
However, Packt Publishing cannot guarantee the accuracy of this information.

First published: August 2016

Production reference: 1120816

Published by Packt Publishing Ltd.
Livery Place
35 Livery Street
Birmingham B3 2PB, UK.

ISBN 978-1-78588-586-0

www.packtpub.com

Credits

Author

Avisekhhar Roy

Project Coordinator

Izzat Contractor

Reviewer

Attilio Carotenuto

Proofreader

Safis Editing

Commissioning Editor

Edward Gordon

Indexer

Tejal Daruwale Soni

Acquisition Editor

Rahul Nair

Graphics

Disha Haria

Content Development Editor

Anish Sukumaran

Production Coordinator

Melwyn Dsa

Technical Editor

Taabish Khan

Cover Work

Melwyn Dsa

Copy Editors

Sonia Mathur

Karuna Narayanan

About the Author

Avissekhar Roy is a B.Tech engineer in computer science. He has had a passion for coding since his school days. However, he had no plans to become a game en, he fell in love with game development.

Avissekhar has worked in many formats of game development environment, ranging from small companies and individual studios to corporate companies and full-scale 2016 and is currently working on games for the mobile platform.

Avissekhar has also worked with some big companies, such as Reliance Games in India, as well as a small-scale studio called Nautilus Mobile. He is now trying to acquire a position in the gaming industry for his own venture, Funboat Games.

I would like to mention my parents, who have supported me in every step during the journey of my career. I would not be able to write this book without their blessings. I would like to thank Mr. Pritesh Dhawle for his active support in writing the book; he is not just my partner at Funboat Games, but also an intimate friend. I'd also like to express my gratitude to Mr. Kinshuk Sunil, who supported me while writing this book at an early stage. There are many more friends and well-wishers whom I would like thank for their support.

Finally, I would like to express my gratitude toward the people who provided their valuable analysis on specific subjects; their articles and reports have helped me a lot to research more while writing this book.

About the Reviewer

Attilio Carotenuto is a senior game designer and developer with over 7 years of experience in his field. He's the owner and game director at Himeki Games, an indie studio with a focus on hardcore, premium games, currently working on *An Oath to the Stars*, a Japanese-style bullet hell shooter.

Attilio previously worked at companies such as Electronic Arts Playfish, King, and every day.

or *Building*

Levels in Unity, Volodymyr Gerasimov; *Unity3D UI Essentials*, Simon Jackson; and *Unity 3D Game Development by Example [Video]*, Adam Maxwell.

You can find more about his recent projects, articles, and talks on his personal website at <http://www.attiliocarotenuto.com/>.

www.PacktPub.com

eBooks, discount offers, and more

ith PDF

and ePub files available? You can upgrade to the eBook version at www.PacktPub.com or purchase a print copy.

Get in touch with us at customercare@packtpub.com for more details.

At www.PacktPub.com, you can also read a collection of free technical articles, sign up for a range of free newsletters and receive exclusive discounts and offers on Packt books and eBooks.



<https://www2.packtpub.com/books/subscription/packtlib>

online digital
library of books.

Why subscribe?

- Fully searchable across every book published by Packt
- Copy and paste, print, and bookmark content
- On demand and accessible via a web browser

Table of Contents

Preface	xvii
Chapter 1: Android Game Development	1
Android game development	1
Features and support	3
Challenges	4
User experience	4
Design constraints	5
A game is not just an application	5
Games versus applications	5
Life cycle of Android application and games	6
Performance of games and applications	7
Memory management of games and applications	7
Choosing the target device configuration	8
Game scale	8
Target audience	9
Feature requirement	9
Scope for portability	10
Best practices for making an Android game	10
Maintaining game quality	11
Minimalistic user interface	11
Supporting maximum resolutions	12
Supporting maximum devices	12
Background behavior	13
Interruption handling	13
Maintaining battery usage	14
Extended support for multiple visual quality	15
Introducing social networking and multiplayer	15
Summary	16

Chapter 2: Introduction to Different Android Platforms	17
Exploring Android mobiles	18
Exploring Android tablets	22
Exploring Android televisions and STBs	24
Exploring Android consoles	28
Exploring Android watches	33
Development insights on Android mobiles	35
Development insights on Android tablets	38
Development insights on Android TV and STBs	39
UI and game design	41
Overscan	41
Development insights on Android consoles	42
Development insights on Android watches	42
Creating and setting up a wearable application	43
Including the correct libraries in the project	44
Hardware compatibility issues with Android versions	44
Platform-specific specialties	44
Android mobiles	45
Android tablets	45
Android televisions and STBs	45
Android consoles	46
Android watches	46
Summary	46
Chapter 3: Different Android Development Tools	49
Android SDK	50
Android Development Tool	50
Android Virtual Device	51
Configuring AVD	51
Android Debug Bridge	53
Using adb on an Android device	54
Dalvik Debug Monitor Server	55
Other tools	56
Eclipse	56
Hierarchy Viewer	57
Draw 9-Patch	58
ProGuard	59
Asset optimization tools	60
Full asset optimization	60
Creating sprites	61

Tools for testing	61
Creating a test case	61
Setting up your test fixture	61
Adding test preconditions	63
Adding test methods to verify an activity	63
Performance profiling tools	64
Android Studio	65
Android project view	65
Memory and CPU monitor	66
Cross-platform tools	67
Cocos2d-x	68
Unity3D	69
Unreal Engine	70
PhoneGap	71
Corona	72
Titanium	73
Summary	74
Chapter 4: Android Development Style and Standards in the Industry	75
The Android programming structure	76
Class formation	76
Call hierarchy	77
Game programming specifications	78
Gameplay programming	78
Graphics programming	79
Technical programming	79
Sound programming	80
Network programming	80
Game tool programming	80
Research and development programming	81
Technical design standards	81
Game analysis	82
Design pattern and flow diagram	82
Technical specification	82
Tools and other requirements	83
Resource analysis	83
Testing requirements	83
Scope analysis	84
Risk analysis	84
Change log	84

Game design standards	85
Game overview	85
Gameplay details	85
Game progression	86
Storyboard and game elements	86
Level design	86
Artificial intelligence	86
Art style	86
Technical reference	87
Change log	87
Other styles and standards	87
Different styles for different development engines	88
Different programming languages	88
Different work principles	88
Different target platforms	89
Industry best practices	89
Design standards	89
Programming standards	90
Testing standards	91
Summary	91
Chapter 5: Understanding the Game Loop and Frame Rate	93
Introduction to the game loop	94
User input	94
Game update	95
State update	96
Rendering frames	96
Creating a sample game loop using the Android SDK	97
Game life cycle	101
Game update and user interface	102
Interrupt handling	106
General idea of a game state machine	107
The FPS system	110
Hardware dependency	112
Display or rendering	113
Memory load/unload operations	113
Heap memory	113
Stack memory	114
Register memory	114
ROM	114
Logical operations	114

Balance between performance and memory	115
Controlling FPS	116
Summary	117
Chapter 6: Improving Performance of 2D/3D Games	119
2D game development constraints	119
2D art assets	120
Sets of 2D art assets	120
Same asset set for multiple resolutions	120
Number of assets drawn on screen	120
Use of font files	121
2D rendering system	122
2D mapping	123
2D physics	125
Box2D	125
LiquidFun	126
Performance impact on games	126
2D collision detection	126
Rectangle collision	127
Rectangle and circle collision	129
Circle and circle collision	131
Performance comparison	132
3D game development constraints	133
Vertices and triangles	133
3D transformation matrix	133
3D object and polygon count	134
3D rendering system	135
3D mesh	135
Materials, shaders, and textures	136
Textures	136
Shaders	137
Materials	137
Collision detection	137
Primitive colliders	137
Mesh colliders	137
Ray casting	138
Concept of "world"	139
Elements of the game world	139
Light sources in the game world	139
Cameras in the game world	143
The rendering pipeline in Android	145
The 2D rendering pipeline	145
The 3D rendering pipeline	146

Optimizing 2D assets	147
Size optimization	147
Data optimization	147
Process optimization	148
Optimizing 3D assets	148
Limiting the polygon count	148
Model optimization	148
Common game development mistakes	149
Use of non-optimized images	149
Use of full utility third-party libraries	149
Use of unmanaged networking connections	149
Using substandard programming	150
Taking a shortcut	150
2D/3D performance comparison	151
Different look and feel	151
3D processing is way heavier than 2D processing	151
Device configuration	152
Processor	152
RAM	152
GPU	153
Display quality	153
Battery capacity	153
Summary	154
Chapter 7: Working with Shaders	155
Introduction to shaders	156
What is a shader?	156
Necessity of shaders	156
Scope of shaders	158
How shaders work	158
Types of shaders	159
Pixel shaders	159
Vertex shaders	159
Geometry shaders	159
Tessellation shaders	159
Android library shaders	160
Writing custom shaders	161
Shaders through OpenGL	163
Use of shaders in games	169
Shaders in a 2D game space	169
Shaders in a 3D game space	170
Summary	173

Chapter 8: Performance and Memory Optimization	175
Fields of optimization in Android games	176
Resource optimization	176
Art optimization	176
Sound optimization	177
Data file optimization	177
Design optimization	177
Game design optimization	177
Technical design optimization	178
Memory optimization	178
Don't create unnecessary objects during runtime	179
Use primitive data types as far as possible	180
Don't use unmanaged static objects	180
Don't create unnecessary classes or interfaces	180
Use the minimum possible abstraction	181
Keep a check on services	181
Optimize bitmaps	181
Release unnecessary memory blocks	182
Use external tools such as zipalign and ProGuard	182
Performance optimization	183
Using minimum objects possible per task	183
Using minimum floating points	184
Using fewer abstraction layers	185
Using enhanced loops wherever possible	185
Avoid getter/setters of variables for internal use	185
Use static final for constants	185
Using minimum possible inner classes	186
Relationship between performance and memory management	186
Memory management in Android	186
Shared application memory	187
Memory allocation and deallocation	187
Application memory distribution	188
Processing segments in Android	188
Application priority	188
Active process	189
Visible process	190
Active services	190
Background process	190
Void process	190
Application services	191
Service life cycle	191
Resource processing	191
Drawable resources	192
Layout resources	192
Color resources	192
Menu resources	192

Tween animation resources	192
Other resources	192
Different memory segments	193
Stack memory	193
Heap memory	194
Register memory	195
Importance of memory optimization	195
Optimizing overall performance	196
Choosing the base resolution	196
Defining the portability range	197
Program structure	197
Managing the database	197
Managing the network connection	198
Increasing the frame rate	198
Importance of performance optimization	198
Common optimization mistakes	199
Programming mistakes	199
Design mistakes	200
Wrong game data structure	200
Using game services incorrectly	200
Best optimization practices	201
Design constraints	201
Development optimization	201
Data structure model	202
Asset-using techniques	202
Art assets	203
Audio assets	203
Other assets	204
Handling cache data	204
Summary	205
Chapter 9: Testing Code and Debugging	207
Android AVDs	207
Name of the AVD	209
AVD resolution	209
AVD display size	210
Android version API level	210
Android target version	210
CPU architecture	210
RAM amount	210
Hardware input options	211

Other options	211
Extended AVD settings	211
Android DDMS	211
Connecting an Android device filesystem	212
Profiling methods	213
Thread information monitoring	213
Heap information monitoring	213
Tracking memory allocation	213
Monitoring and managing network traffic	214
Tracking log information using Logcat	214
Emulating device operations	214
Android device testing and debugging	215
Device testing	215
Prototype testing	216
Full or complete testing	216
Regression testing	216
Release testing or run testing	216
Device debugging	217
Use of breakpoints	217
Monitoring the memory footprint	217
Checking log messages	218
Dalvik message log	218
ART message log	218
Checking heap updates	219
Tracking memory allocation	220
Checking overall memory usage	221
Private RAM	221
Proportional set size (PSS)	221
Tracking memory leaks	222
Strategic placement of different debug statements	222
Memory allocation	222
Tracking the object state at runtime	223
Checking the program flow	223
Tracking object values	223
Exception handling in Android games	224
Syntax	224
Scope	226
Null pointer exceptions	226
Index out of bound exceptions	227
Arithmetic exceptions	228
Input/output exceptions	228
Network exceptions	229
Custom exceptions	229

Debugging for Android while working with cross-platform engines	230
Best testing practices	230
Tools and APIs	230
Testing techniques	231
Local test	231
Instrumented test	232
Summary	232
Chapter 10: Scope for Android in VR Games	233
Understanding VR	234
Evolution of VR	234
Modern VR systems	235
Use of VR	235
Video games	235
Education and learning	236
Architectural design	236
Fine arts	236
Urban design	236
Motion pictures	236
Medical therapy	237
VR in Android games	237
History of Android VR games	237
Technical specifications	237
Current Android VR game industry	238
Future of Android in VR	238
Google Daydream	238
Game development for VR devices	239
VR game design	239
VR target audience	239
VR game development constraints	240
Introduction to the Cardboard SDK	240
Cardboard headset components	241
Cardboard application working principle	241
Upgrades and variations	241
Basic guide to develop games with the Cardboard SDK	242
Launching and exiting the VR game	242
Hitting the Back button	242
Hitting the Home button	243
VR device adaptation	243
Display properties	243
In-game components	243
Game controls	244
Control concepts	244

VR game development through Google VR	246
Google VR using the Android SDK	246
Google VR using Android NDK	248
Android VR development best practices	248
Draw call limitations	248
Triangle count limitations	249
Keeping a steady FPS	249
Overcoming overheating problems	249
Better audio experience	250
Setting up proper project settings	250
Using a proper test environment	250
Challenges with the Android VR game market	250
Low target audience	251
Limited game genres	251
Long game sessions	251
Limited device support	251
Real-time constraints	252
Expanded VR gaming concepts and development	252
Summary	253
Chapter 11: Android Game Development Using C++ and OpenGL	255
Introduction to the Android NDK	256
How the NDK works	256
Native shared library	256
Native static library	257
Build dependency	257
Android SDK	257
C++ compiler	257
Python	258
Gradle	258
Cygwin	258
Java	258
Native project build configuration	258
Android.mk configuration	258
Application.mk configuration	260
C++ for games – pros and cons	261
Advantages of using C++	261
Universal game programming language	261
Cross-platform portability	261
Faster execution	262
CPU architecture support	262
Disadvantages of using C++	262
High program complexity	262

Platform-dependent compiler	263
Manual memory management	263
Conclusion	263
Native code performance	264
Rendering using OpenGL	265
OpenGL versions	265
OpenGL 1.x	265
OpenGL 2.0	265
OpenGL 3.0	266
OpenGL 3.1	266
Detecting and setting the OpenGL version	266
Texture compression and OpenGL	267
ATC	267
PVRTC	267
DXTC	267
OpenGL manifest configuration	268
Choosing the target OpenGL ES version	269
Performance	269
Texture support	269
Device support	269
Rendering feature	270
Programming comfort	270
Different CPU architecture support	270
Available CPU architectures	270
ARM	270
x86	271
Neon	271
MIPS	271
Advantages and disadvantages of integrating multiple architecture support	271
Summary	272
Chapter 12: Polishing Android Games	273
Requirements for polishing	274
Development polishing	274
Memory optimization	274
Performance optimization	274
Portability	275
Art polishing	275
UI polishing	275
Animation polishing	275
Marketing graphics	275
Design polishing	276
Designing UX	276
Polishing the game flow	276
Polishing the metagame	276

Game economy balance	276
Game difficulty balance	277
Play testing	277
User gameplay difficulty levels	277
User actions during gameplay	278
User actions while browsing the game	278
Whether the user is paying or not	278
Whether the game is running smoothly	279
Whether the user can adopt the gameplay	279
User retention	280
Taking care of the UX	280
Visual effects	280
Sound effects	281
Theme music	281
SFXs	281
Transaction effects	281
Action feedback	281
Android-specific polishing	282
Optimum use of hardware buttons	282
Sticking to basic Android features and functionalities	282
Longer background running	283
Following Google guidelines for Play Store efficiency	283
Game portability	283
Support for various screen sizes	283
Support for multiple resolutions	284
Support for multiple hardware configurations	284
Summary	285
Chapter 13: Third-Party Integration, Monetization, and Services	287
Google Play Services	288
Google Analytics	288
Significance	288
Integration tips	289
Best utilization	289
Google IAB	289
The Google IAB model	289
Integrating Google IAB	290
Advantages and disadvantages of Google IAB	290
Google Leaderboard	291
Significance	291
Integrating Google Leaderboard	291
Variety of leaderboards	292
Options for storing and displaying leaderboards	292

Push notifications	293
Database	293
Server	293
Target device	293
GCM service	294
Integrating push notifications	295
Significance of push notifications	298
Multiplayer implementation	299
Real-time multiplayer	299
Turn-based multiplayer	300
Single-screen real-time multiplayer	301
Pass and play turn-based multiplayer	301
Local network multiplayer	302
Analytic tools	302
Requirement of analytics tools	302
User behavior	303
Game crash reports	303
Game event triggers	303
Gameplay session timing	303
Gameplay frequency	303
Game balancing	303
User retention	304
Piracy prevention	304
Monetization aspects of analytic tools	304
Identify popular regions of the game	304
Identify a user's likes and dislikes	305
Validate and improve the metagame	305
Track paying users	305
Track and count advertisement display	305
Some useful analytic tools	305
Flurry	306
GameAnalytics	306
Crashlytics	306
AppsFlyer	306
Apsalar	306
Mixpanel	306
Localytics	307
Appcelerator	307
Android in-app purchase integration	307
What are in-app purchases?	307
In-app purchase options	308
Store billing services	308
Career billing services	309
Types of in-app purchases	310
Consumable items	310
Non-consumable items	310
Subscriptions	310

Android in-game advertisements	311
Requirement for advertisements	311
Terminologies in advertisement monetization	312
eCPM	312
CPC/CPA	312
CPI	312
RPM	312
Fillrate	313
Types of advertisements	313
Banner advertisements	313
Interstitial advertisements	314
Video advertisements	315
In-game dynamic advertisements	315
Monetization techniques	315
Premium model	316
Free model	316
Freemium model	316
Try-and-buy model	316
Planning game revenue	316
Revenue versus profit	317
Revenue sources	317
Advertisement revenue	317
In-app purchase revenue	317
Other revenue sources	318
Regional variations of revenue plan	318
User base variations	319
User behavior variations	319
User acquisition techniques	319
Game promotion channels	320
YouTube channels	320
Android forums	320
Sports forums	320
Facebook promotion	321
Twitter and other social platforms	321
Game blogs and forum discussions	321
Paid user acquisition	321
Other techniques	322
User retention techniques	322
Daily bonus	323
Leaderboards and achievements	323
Offerwall Integration	323
Push notifications	323
Frequent updates	324

Table of Contents

Featuring Android games	324
Creativity and uniqueness	324
User reviews and ratings	324
Download count	325
Revenue amount	325
Publishing Android games	325
Self publishing	325
Publishing through publishers	326
Summary	326
Index	327

Preface

Fun is the keyword that creates the necessity for entertainment in life. There are
rms.
es when
gaming was limited to sports, board games, card games, and the like. Then, games
entered the digital domain with specific gaming devices. Gradually, they have come
to the mobile platform now. Android is one of the most promising platforms. The
Android market is growing each day and Android gaming is growing with it.

This book is mainly aimed at game programmers. Many people consider game

de editor
and typing in a computer language, it's about creating a medium of spreading
entertainment.

lly for
gramming
this book with my
experiences throughout my career so far.

What this book covers

Chapter 1, Android Game Development, will introduce you to the guidelines and rules
of game development on the Android platform.

Chapter 2, Introduction to Different Android Platforms, will disclose the current variants
of Android devices, such as smartphones, TVs, tablets, and smartwatches. It will
elaborate all the possible difficulties while creating a game on these platforms and
the possible solutions.

Chapter 3, Different Android Development Tools, will expose the different tools available to develop an Android application and how to choose suitable tools for specific purposes.

Chapter 4, Android Development Style and Standards in the Industry, will cover the current development style and standards in the game development domain. This is the old SDK.

Chapter 5, Understanding the Game Loop and Frame Rate, will demonstrate the creation and maintenance of game loop using the Android SDK (Java). This chapter will also cover the effects of game loop on the frame rate.

Chapter 6, Improving Performance of 2D/3D Games, will explain all the constraints of 2D and 3D game development on Android, along with the common mistakes and ways to avoid them in order to improve performance.

Chapter 7, Working with Shaders, will describe the use of shaders on the Android game development.

Chapter 8, Performance and Memory Optimization, will provide in-depth knowledge of optimizing any Android game.

Chapter 9, Testing Code and Debugging, will teach you the different ways to debug an Android game.

Chapter 10, Scope for Android in VR Games, will introduce you to virtual reality for game development on Android. This chapter describes various scopes of VR and its future in game development.

Chapter 11, Android Game Development Using C++ and OpenGL, will briefly explain game development using C++ and OpenGL.

Chapter 12, Polishing Android Games, will focus on the completion of an Android game and make it ready for release.

Chapter 13, Third-Party Integration, Monetization, and Services, will elaborate the possible integration of any third-party tools or SDKs in order to monetize the game.

What you need for this book

It is assumed that the reader is already a game developer who has worked on the Android platform. You need to have a clear idea about Android programming using Java and C++.

The reader needs to work on various Android development platforms; most of the code works with the Android SDK. You also need to know the concept of several and more.

Who this book is for

ping
games for Android. A good understanding of game development and basic
will be appreciated.

Conventions

In this book, you will find a number of text styles that distinguish between different kinds of information. Here are some examples of these styles and an explanation of their meaning.

Code words in text, database table names, folder names, filenames, file extensions, pathnames, dummy URLs, user input, and Twitter handles are shown as follows:
"We can include other contexts through the use of the `include` directive."


A block of code is set as follows:


```
<application
<!-- other declarations and tags -->
android:isGame="true"
<!-- other declarations and tags -->
>
```

Any command-line input or output is written as follows:

```
cd platform-tools
```

New terms and **important words** are shown in bold. Words that you see on the his: "In the
Configure Project window, enter a name for the application."

[ Warnings or important notes appear in a box like this.]

[ Tips and tricks appear like this.]

Reader feedback

Feedback from our readers is always welcome. Let us know what you think about our titles as it helps us develop titles that you will really get the most out of.

To send us general feedback, simply e-mail feedback@packtpub.com, and mention the book's title in the subject of your message.

If there is a topic that you have expertise in and you are interested in either writing or contributing to a book, see our author guide at www.packtpub.com/authors.

Customer support

Now that you are the proud owner of a Packt book, we have a number of things to help you to get the most from your purchase.

Errata

Mistakes do happen. If you find a mistake in one of our books – maybe a mistake in the text or a figure, you can save other readers from frustration and help us improve subsequent versions of this book. If you find any errata, please report them by visiting <http://www.packtpub.com/submit-errata>, selecting your book, clicking on the **Errata Submission Form** link, and entering the details of your errata. Once your errata are verified, your contribution will be added to any list of existing errata under the Errata section of that title.

To view the previously submitted errata, go to <https://www.packtpub.com/books/content/support> and enter the name of the book in the search field. The required information will appear under the **Errata** section.

Piracy

oss all
ry seriously.

If you come across any illegal copies of our works in any form on the Internet, please provide us with the location address or website name immediately so that we can pursue a remedy.

Please contact us at copyright@packtpub.com with a link to the suspected pirated material.

We appreciate your help in protecting our authors and our ability to bring you valuable content.

Questions

t
questions@packtpub.com, and we will do our best to address the problem.

1

Android Game Development

ecade.

Previously, it was limited to PCs, consoles, and a few embedded gaming devices.

gy, better

portability, better flexibility, and better quality. This has opened up the doors for developers to create games with better quality and fewer limitations.

Android is a modern age operating system, and is being used widely for many hardware platforms. Hence, the world of Android has become a target for game developers. The most efficient and useful targets are Android smartphones and Android tops

is being used

in TVs and smart watches also. Hence, the popularity of Android is touching the sky among game developers.

roid game

development. Let's start with the following topics:

- Android game development
- A game is not just an application
- Choosing target device configuration for your game
- Best practices while making a game on Android

Android game development

Let us now focus on the main topic of this book. Although game development covers many platforms and technologies, we will only focus on Android in this book.

Android is a mobile operating system based on the Linux kernel. Currently, it is to date.

read), this

OS caught the attention of many game developers. Android uses what is called the **Dalvik Virtual Machine (DVM)**, which is an open source implementation of a **Java Virtual Machine (JVM)**. There are several differences between Dalvik and a standard JVM, some subtle, some not so subtle. The DVM is also not aligned to either Java SE All of

this makes for a slight learning curve if you happen to be transitioning from Java ME. Google introduced an alternative to DVM called **Android RunTime (ART)** from Android 4.4 (KitKat), and ART replaced DVM from Android 5.0 (Lollipop). ART mainly features **Ahead-of-time (AOT)** compilation, and an improved garbage collection process, and it provides a smaller memory footprint in order to optimize memory operations. However, most game developers use DVM to support older versions of Android devices.

Android game development started extensively when this OS was adapted by many hardware platforms. Android is mostly being used on the mobile and tablet platforms. When the mobile game industry started migrating from Symbian or Java to Android or other smart mobile OSes, Android game development started to boom.

There are a few reasons for the success of Android games:

- Smooth user interface
- Better interactivity
- Touch interface
- Better look and feel
- Better hardware platform
- More design flexibility

It is always easier to use a common operating system than an embedded **real-time operating system (RTOS)**. The user need not spend time on different hardware to learn its usability. Android is one such easy-to-use operating system.

ns on better

hardware configuration than Symbian, Java, or an embedded OS. It enhances user ganizations.

As the user base of Android increased, many more game developers started targeting this platform.

gave

flas

enhanced.

rt from mobile phones, Android is being used on tablets, televisions, wristwatches, consoles, digital cameras, PCs, and other devices. Nowadays, game developers are targeting almost every Android platform.

Features and support

Direct manipulation interface is the top feature of Android. It interacts with the user -time action, and dynamic feedback. Android mainly uses a touch interface with real-time action ed in game development for Android.

Android application development is mainly based on Java (SDK) and C++ (NDK), which are the most common programming languages in the world. Hence, developing a game has become much easier.

Excellent support for multimedia took Android a step further in gaining popularity. Game developers can now use multimedia objects freely inside the game in order to increase the game quality.

called Google Play Services. It is a closed system-level API service provider, which has proved to be very useful in game development.

ve also eased the job of game developers. Some of the tools we can mention are Android Studio, App Inventor, Corona, Delphi, Testdroid, Sample Directmedia Layer, Visual Studio, Eclipse IDE, and RubyMotions.

Android device hardware configuration has to follow a minimum configuration list, so it becomes very easy for the developers to identify the configuration. Moreover, it has to maintain a minimum standard to run applications easily.

There are plenty of sensors associated with Android devices (mostly on mobiles or game.

Android supports awesome connectivity through Bluetooth, Wi-Fi, GSM/CDMA/EDGE, LTE, NFC, IDEN, and the like. These help game developers to create multiplayer games easily.

Virtual reality is another field where Android is being used through Cardboard SDK. We will discuss this topic more later on.

st of the
ys a
chance to explore more, and create a few specific-feature oriented games.

Challenges

The main challenge in developing a game on an Android platform is to make the most use of the features in an efficient way.

The range of Android device configurations is wide. So, designing a game targeting most of them is a big challenge.

Many of the Android game developers design and build games for specific hardware configurations, like Tegra, or Snapdragon, or a particular device like Xperia Play. Nvidia's Tegra is the most commonly used chip in these situations; the THDAs

brought
e use of

Tegra-specific APIs to build their games. The problem with this scenario is that most users don't have Tegra in their phones. In fact, many LTE handsets that might have on S4.

Now, for a developer, it is very difficult to maintain performance across different graphic processors.

User experience

Android games can provide awesome user experience through their features.

Game controls can use the accelerometer or gravity sensor for a physics-based mechanism (if supported by the hardware), which is always an added advantage for real-time interactivity.

On-touch screen devices, and dynamic controls like swiping, dragging, pinching, and multi-touch, can be experienced through Android.

visual
quality of the game.

Miracast in Android is another feature which enables games to use multiple displays and screen sharing for a better experience.

Design constraints

Development of any game requires a design. Android is not an exception. The design
There are
thousands of varieties available for Android. Designers have to choose their target
very carefully, and then design the game scope.

for the
designers as well. Different Android devices have different configurations, but it is
d.

A game is not just an application

ame
development and vice versa. Many do not change their style, and approach game
development accordingly. Every developer of games should keep in mind that *a
game is not just an application.*

Games versus applications

main
ise. On
ier with
a mechanical job. So the development approaches for these two are completely
different. However, this still remains a point of discussion, as every game is an
provide a
better user experience.

It is diffi
versus an application. However, game development has an edge. Most of the
application developers do not have to focus much on speed performance, whereas
all game developers have to focus on speed and the frame rate of the game.

me. This
ame
l the
features of an application.

is
fun-oriented. This increases the difficulties in game development. Fun is an emotion,
oper can
never know what exactly the game is going to achieve in terms of fun. On the other
target can
be achieved if all the specifications meet the requirement.

s

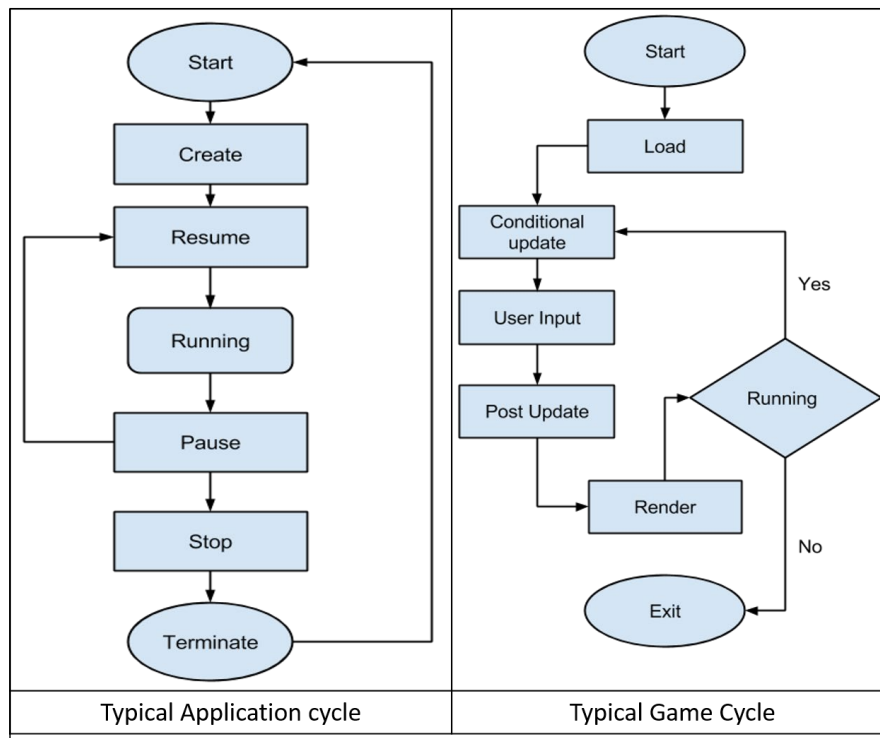
or graphics side; even AI needs a lot of mathematics for the low-level stuff. Applications are more technology driven, with limited use of graphics.

Any application that qualifies as a game must fulfill the following criteria:

- It must entertain a set of users in terms of fun
- There must be a set of milestones to achieve for the users of the application
- It should reward the users for achieving a milestone
- It should have a more dynamic user interface
- There must be better visual impact
- It should be performance driven rather than feature driven

Life cycle of Android application and games

. But a game has more to the cycle, as you can see in the following diagram:



runs within the

e **game loop**.

This will be discussed later in detail.

nnng

state, but there are multiple game update states. In a typical system of game

e game

ne actually

controls the **frame rate**.

Performance of games and applications

ems for

games and applications. Performance is one of the biggest requirements in game development, whereas it is only a recommended feature for an application, as the frame rate does not affect the quality.

same scale.

A game runs on repetitive frames — one set of tasks runs on one frame. This increases the instruction traffic for the processor. In an application, there are generally no

, the processor

gets plenty of time to execute the instruction as no instructions are being sent repetitively.

Memory management of games and applications

ames.

In case of games, multimedia assets are the main objects, which occupy a larger

it is just the

opposite. Applications need to load only the object they require for the state, that is, class objects.

For any game developer, memory optimization is a must. Because of the extensive use of memory, a developer cannot afford to have unused objects loaded in memory, or any memory leakage caused by mishandled memory pointers. This has a direct

ously a

on running the

ut

memory optimization.

Choosing the target device configuration

As mentioned earlier, Android has a variety of device configurations. So, it is very important for an Android game developer to choose the target very carefully. The general approach should have these parameters:

- Game scale
- Target audience
- Feature requirement
- Scope for portability

Game scale

This is basically the scale on which the game is being made. The larger the scale, the better the configuration that it'll need. This includes mainly the game size, which means the amount of memory it will consume on a device. Many Android devices are configured with very low RAM and internal memory storage. If the targeted device does not have the required configuration, the game will not run. Even if the game is fully optimized, it can fail depending on the hardware platform it is running on.

requires

processor speed. If a game is process-heavy, and the targeted device has a slow processor, the game will experience some horrible frame rate issue, or crash.

Every Android game developer must be aware of the requirements of memory, processor, and other constraints when choosing the target device.

of disk

space to install, 512 MB of RAM to run, and a 1.2 GHz processor speed to achieve a decent frame rate. Now consider a mobile device which matches these specifications exactly, but being a developer, one must not assume that the device will not have

, there is a fair

device

ust have a

higher configuration than the minimum requirement.

f RAM,

and a 2 GHz multicore processor. There is no doubt that the game will run on that device with maximum performance, but the device could have supported a larger-scale game. So the resource utilization is not efficient in this scenario. This is where porting comes in. A game developer should upscale the game quality, and create a different build for those high-end configuration devices.

It is a very common practice in the industry to exclude a few devices from the targeted device list to make the game run properly. In a few cases, the game developer creates separate game builds to support most of the devices and maintain the game quality.

Target audience

It is assumed
that they will
play the game more than other people.

The direct
audience is
created
because this
specific audience will mostly use mobile devices, as they have less time to sit in front of a television set. So, the list of target devices should contain mobile devices for this target audience.

Now, let's
take the example of Android mobile phones, as this is the most-used Android category. We can see a range of Android devices available in the market. Most of the Android phones are comparatively cheaper, and have fewer features. A major market is in Asia
and, therefore, the
developer should consider the minimum configuration target.

Feature requirement

About
games on Android, the major focus is on mobile and tablet platforms. Mostly,
Android games are made for these devices.

If we consider other platforms like watches, TVs, or consoles, the feature set varies. Televisions provide a bigger display with less user control, watches have limited display area and minimum configuration, consoles have better graphic quality with dedicated controls, and so on. It is very important to identify the feature list which is required to recognize the hardware devices.

Some other
features must have
those features. However, common mobile and tablet devices have almost the same dependency
and becomes very specific when Android games are made for some particular hardware platform like consoles or VR devices.

Scope for portability

sider the
the effort
required to select or choose the target hardware.

per.

porting.

We will only focus on hardware porting here, as we have already fixed the platform to be Android.

A game developer should focus on the following points to increase the portability of a game:

- Creating different sets of assets
- Designing different sets of controls
- Finding and listing alternatives for a feature
- Controlling memory usage
- Controlling the frame rate

points.

Most of the time, the target hardware is chosen first depending on the other the game.

Best practices for making an Android game

ht way

through which the game looks great, and performs well across as many devices as owing points:

- Maintaining game quality
- Minimalistic user interface
- Supporting maximum resolutions
- Supporting maximum devices
- Background behavior
- Interruption handling
- Maintaining battery usage
- Extended support for multiple visual qualities
- Introducing social networking and multiplayer

ater as the book
progresses.

Maintaining game quality

There are millions of games available in the market, and thousands being introduced
veloper
r improved
games.

The developer should keep a constant eye on the reviews and complaints from the
one can
predict the exact user reaction to the game before it is out in the market. So, in most
gn, or other
means, to keep the consumer happy.

here are
several tools available to do this job efficiently, such as Google Analytics, Game
comments on
stores or blogs are helpful to maintain the quality of a game.

Fixing bugs in a game is another major factor in increasing the quality of the game.
elopment.
NRs when
id error
s feature in
Android versions 2.2 and later.

play, and
consistent frame rate.

Minimalistic user interface

This is a typical design practice for Android games. A common mistake that many
take the user
should
experience the game with minimum effort the very first time. Most users leave games
because of the heavy UI interface.

Technically, a developer should take care of the device UI options like Menu, Back,
and Home. These are the most common options for the Android mobile and tablet
the game,
Also, there
should be a quick interface to quit the game.

Basically, having a minimum user interface and fewer screen transactions saves a lot of time, which has a direct impact on gameplay sessions.

Supporting maximum resolutions

This is a very obvious point for creating a good Android game. A game must support as many resolutions as possible. Android, in particular has many different screen sizes available in the market.

Android has a series of different resolution sets:

- LDPI (approximately 120 dpi)
- MDPI (approximately 160 dpi)
- HDPI (approximately 240 dpi)
- XHDPI (approximately 320 dpi)
- XXHDI (approximately 480 dpi)
- XXXHDPI (approximately 640 dpi)

If they do not follow multiple resolution specifications, the developer can also opt for the screen compatibility option available as a last resort. However, it is recommended not to use this feature of Android, because it can reduce the visual quality significantly. This option is, by default, disabled from Android API version 11.

Supporting maximum devices

Other than the different screen sizes, Android has a variety of device configurations. Most developers filter the device list only by screen resolution, which is a bad practice. An Android game developer should always consider the target device configuration along with the resolution.

When building their applications, developers should remember not to make assumptions about specific keyboard layouts, the touch interface, or other interactive systems unless, of course, the game is restricted so that it can only be used on those devices.

Optimizing the application in terms of memory and performance is also helpful in supporting more devices. The developer should not restrict them to only a few sets of devices. Optimal use of disk space and the processor opens up the opportunity to increase the support range.

A single game application build can support more devices with some simple tricks. tion, and nd processing speed can be controlled.

Background behavior

running. These are called asynchronous tasks, mostly used for loading a large file or fetching something from the Internet.

Another type of background task is called services, which works even when the main communicating with the device on which the game is installed.

e game properly. A large chunk of data usually takes longer time, but it should not pause game to keep the main thread running, and provides dynamic feedback.

Background services are useful for increasing the communication between the developer and user. They can provide user activity information to improve the game as well as notifying users about the latest update or information.

Interruption handling

As we discussed earlier about the game loop, the loop pauses or, sometimes, terminates on any external interruption. In an ongoing game cycle, the interruption should not hat the game restarts after being interrupted. Android is most likely to kill the game activity y needs provision ess.

It is good practice to save the user progress periodically to avoid any loss of data or progression. But saving data may cause lags in the game loop, and can drop the frame rate significantly. The game developer should identify the states where the data can be saved without affecting the gaming experience.

t and pause/
ning the
roperly on
interruption. In most cases, all of the background processes do not pause, causing
unusual behavior by the game.

Maintaining battery usage

One of the reasons for the success of an Android game is power efficiency. Most
a limited
source of power. So power-saving applications are always preferred.

tivity.
ry. So,
there is a fair chance that the game uses up a lot of power.

Most game developers focus a lot on visual appearance. It increases the graphic
r the developer
ssets should
not boost up processing or rendering, as, developers often use non-optimized assets.

These are
used widely for better connectivity with consumers or for some web-based services.
ired network.
illing a
service which is not connected for a long time or was disconnected from the network,
with the help of Android **PackageManager**.

ser count than
another, better-quality game, just because of lower battery consumption.

he receivers
s.
ing
online
rring
update alarm.

higher
bandwidth to complete simply by enabling a broadcast receiver, which will listen
cation is
connected to Wi-Fi. This significantly reduces battery use.

Extended support for multiple visual quality

This section actually starts with supporting multiple resolutions. We have already discussed multiple-size screens with different dpi. The following list is another standard that Android devices follow:

- QVGA (low PPI)
- WQVGA (medium-low PPI)
- HVGA (medium-high PPI)
- WVGA (medium-high PPI)
- SVGA (high PPI)
- VGA (very high PPI)

Creating graphics using this standard is always beneficial in order to achieve the best possible visual quality across devices. This notation mainly depends on the screen size, irrespective of the resolution. It is very common for Android devices to use this standard to ensure visual quality.

Introducing social networking and multiplayer

Introducing social networking is a real user on a large user base and retention rate significantly.

to experience the same game state together, and to improve the game play by real-time interaction. A few board games such as Chess, Ludo, and Snakes and Ladders, are examples of such a possibility. Beside those, some real-time online multiplayer games are also at their peak.

sides introduced through Wi-Fi, called **Google Nearby**. There are many other third-party platforms that support multiplayer.

Summary

Making an Android game is not difficult, making a successful game is. From a technical point of view, a successful game must provide smooth gameplay to provide users with an excellent, swift gaming experience. Great visual quality with better graphics always attracts users and other potential players nearby, while fewer perform

according to plan. A wide range of device support can increase the number of users and gameplay sessions, optimal use of resources ensures the minimum possible application package size, and finally, a good relationship between the developer doubts and confusions of the users.

ow to make

a successful Android game. Making an Android game is no different to making any software. However, a game must follow some practices in order to achieve its fun element. You will learn in detail about making an efficient Android game later in this opment for f available

Android devices made by various manufacturers. There are many types of devices, which we will have a look into.

We will try to explore a better and efficient approach for Android game development ,with many development procedures, styles, and standards for different hardware platforms. We will further dig deep, with game-specific development standards for pment, with shaders and various optimization techniques.

Then, finally, we will explore various ways to make a successful game, which is good bout e power of data collected from users through this book.

2

Introduction to Different Android Platforms

The first commercially released Android device was the **HTC Dream**. In 2008, this
ce
then, thousands of manufacturers have been using Android for their devices. At
first, Android became popular among mobile operating systems such as Symbian,
a new,
filled these
ompeting
operating systems.

According to the latest market study, in the first quarter of 2016, Android holds
sage of time,
platforms
like tablets, televisions, watches, consoles, and so on.

game
:

- Exploring Android mobiles
- Exploring Android tablets
- Exploring Android televisions
- Exploring Android consoles
- Exploring Android watches
- Development insight on Android mobiles
- Development insight on Android tablets
- Development insight on Android television and STB

- Development insight on Android consoles
- Development insight on Android watches
- Each platform has its own specialty
- Going cross-platform for the same game
- Required limitation measurement before design

velop games.

The modern world has witnessed that games are now not just limited to PCs or consoles. They have become a part of almost everything. So, it is very important for Android game developers to have a decent knowledge of all possible hardware that e.

Exploring Android mobiles

Android mobile devices are the most important devices for game developers. Mobile the black le competitor, the iPhone.

Initial Android gaming got its momentum after the release of Android version 1.6, followed by Android 2.3. Even today, there are many devices running on Android rt Android 2.3.

There was a time when Android used to run with a minimum requirement of 32 MB of RAM, 32 MB of disk space, and a 200 MHz processor as well. If we take a look at current device specifications, a drastic change can be noticed. Nowadays, Android mobile devices have 1 GB RAM, 1 GHz processor, and 4 GB disk space on an average. Most of the devices have multicore processing units. However, this t increased the complexity even more.

Let's have a look at the specifications of a low-budget Android device with a comparatively low configuration. The following example table shows the configuration of a Micromax Bolt A24:

Processor	Cortex A5
Speed	1 GHz
RAM	256 MB
Flash memory	512 MB

Screen mode	NA
Screen resolution	480x640
Screen size	2.8 inch
Android version	2.3 (Gingerbread)

Here is what it looks like (image source: <http://www.androided.in/wp-content/uploads/2014/02/Micromax-BoltA24.jpg>):



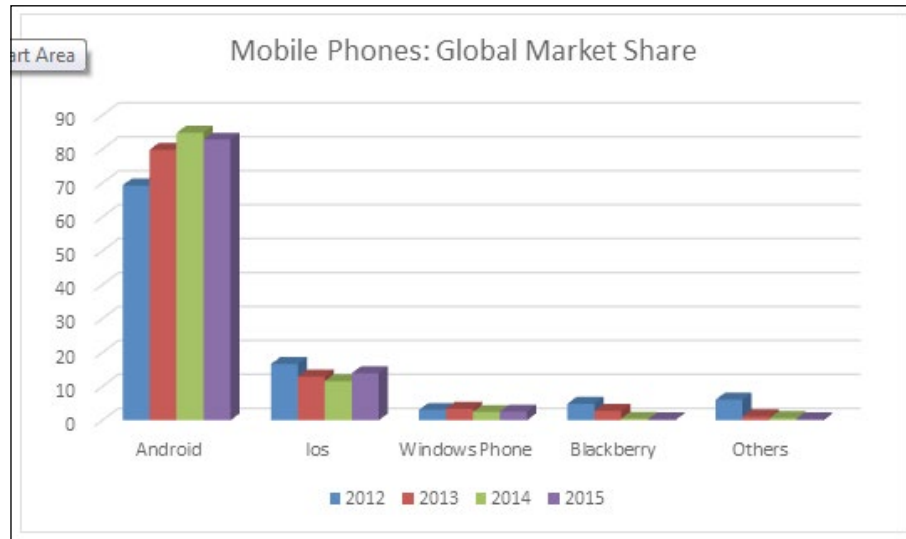
Now take a look at a very high-budget Android device with a very high configuration. The following table shows the configuration of a Samsung Galaxy S6:

Processor	Cortex A57
Speed	2.1 GHz quadcore
RAM	3 GB
Flash memory	128 GB
Screen mode	NA
Screen resolution	1440x2560
Screen size	5.1 inches
Android version	5.0.2 (Lollipop)

Here is what it looks like (image source: <http://talishop.ru/data/big/ew.jpg>):



Every Android game developer should be well aware of the fact that a single game build cannot achieve best performance across all configurations. It is pretty t will surely an the quality of the game on Samsung Galaxy S6 would be far below expectations. So it is ill discuss this in detail later on.



OS trails
behind with a huge gap in between.

The rest of the mobile operating systems such as Windows Phone, BlackBerry, Java, cited that
the future mobile phone market will be dominated by Android and iOS.

day,
around tens of thousands of new games are being launched for Android mobiles on stores like Google Play Store, Amazon App Store, a few career-specific stores, and many more individual online sites.

exploring
Android mobile devices, we need to understand the user group. In most cases, the user category is recognized by the device configuration or the price group.

Exploring Android tablets

Android tablets are very similar to Android mobile phones. The main specification is size of a mobile.

However, there is no hard-and-fast rule or any specific system to measure that.

The minimum requirement to run Android was a 200 MHz processor, 32 MB RAM, and 32 MB disk space. Android requires an ARMv5 or higher processor, although Android 4 requires ARMv7. Previously, starting age tablets had almost the minimum hardware system.

touchscreen tablets are typically larger than phones or PDAs.

One of the minimum configured Android tablet is Coby Kyros MID7047, which is shown here (image source: http://www.evisionstore.com/catalogo/coby_mid7012-4g.jpg):



The following table shows its specifications:

Processor	Cortex A5
Speed	1 GHz
RAM	512 MB
Flash memory	4 GB
Screen mode	WVGA
Screen resolution	800x480
Screen size	7.00 inch
Android version	4.0 (ICS)

ore than

121 million devices. To add more to the number, the Amazon Kindle Fire sold about 7 million devices.

As the volume of Android tablets has increased, the configuration is getting better. Let's have a look at a tablet configuration released in June 2015 by Sony. The model is called the Sony Xperia Z4 (image source: <https://tecneticocl.wpengine.com/wp-content/uploads/2015/03/z4tab.png>):



The following table shows its specifications:

Processor	Snapdragon 810
Speed	2 GHz octacore
RAM	3 GB
Flash memory	32 GB
Screen resolution	2560x1600
Screen size	10.1 inch
Android version	Android 5.0 (Lollipop)

We can clearly observe the huge configuration difference between the two tablets. The funny fact is that both the configurations coexist in the market even today, and people are buying them.

also
have better battery capacity. So games for tablets should be energy efficient. However, modern age tablets do not differ much from smartphones. The size of the smartphone is getting bigger, and smartphone features are being added to tablets. Now, one can even use a tablet as a phone.

Exploring Android televisions and STBs

After the success of Android mobiles and tablets, Android started expanding its target, as

it became

interactive through Android. Google has released a few additional accessories for Android televisions to increase the user experience.

The first Android television device launched for consumers was the Nexus Player on November 3, 2014. The specification of this is as follows:

- Intel Atom Z3560 1.8 GHz quadcore processor
- 1 GB RAM
- 8 GB Flash storage
- Android 5.1.1 Lollipop

Here's what it looks like (image source: http://1.bp.blogspot.com/-H1rp1fboU6g/VX2Huu6ClyI/AAAAAAAA3hA/5te4NZ65Tgg/s1600/nexus_player.jpg):



This was the first kind of Android television device, very similar to Google TV and comparable with Apple TV also.

Apart from Nexus Player, there are a few more Android-enabled STB devices:

- **Freebox Mini 4K:** This is a 4K capable TV STB, originally offered by a French ISP with fiber modem.
- **Forge TV:** This is a television/microconsole with high hardware specs, Snapdragon 805 processor, 2 GB RAM and 16 GB of Flash storage.
- **Shield Android TV:** This device was announced by NVIDIA on March 3, 2015. This claims to give tough competition to the eighth-generation consoles. This hardware set comes with an NVIDIA-branded game controller.
- **OgleBox Android TV:** This device mainly provides region-based content in Australia. It was announced in March 2015.
- **LG UPlus Android TV:** The Korean telecoms company LG UPlus introduced an Android TV on U+ tvG 4K UHD and U+ tvG Woofer IPTV STBs.
- **Arcadyan BouygtelTV:** In June 2015, a French telecom company, Bouygues Telecom, announced an integrated STB codenamed "Miami" based on Android TV.

Besides these STB/console-based Android devices, many television manufacturers are targeting the launch of Android television sets. Sony, Sharp, Philips, LG, Samsung, and other companies are migrating to Android. Philips has announced

(image source: <http://www.techdigest.tv/wp-content/uploads/2014/07/kogan-smart-tv-may-2013.jpg>):



Let's take a look at the specifications of an example smart TV:

- **Model:** VU 32K160M LED TV
- **Operating system:** Android v4.4 KitKat
- **Processor:** 2.0 GHz octacore GPU with Amlogic quadcore S802 ARM Cortex processor CPU
- **Display:** 32" LED screen (1366x768 px) with achromatic technology and full color optimizer, which provides a world class viewing experience for images, videos, games, and so on
- **Design:** The model is covered in A+ grade pure prism panel which makes pictures more sharp and detailed. And it is surprisingly thinner than the common smart TV models. Its dimensions are 29 inch x 19 inch x 7 inch, and it weighs 7.3 kg.
- **RAM and storage:** The Vu 32K160M TV has 2 GB DDR3 RAM and 8 GB NAND Flash storage.

- **Video:** It plays 1080p videos @60fps, and supports various file formats.
- **Connectivity:** Vu TV supports Bluetooth v4.0, Wi-Fi, and Ethernet. It lets the user browse their favorite sites and check mails from the TV screen. It has three USB ports and two HDMI ports.

This specification is good enough to attract a game developer to get active with his/her next game.

Google has also released an Android TV development kit called ADT-1. This is the I/O

(image source: <http://chezasite.com/media/2014-post-icon/adt-1-android-tv-reference-design-1920x1080.jpg>):



One of the things that early Google TV devices were criticized for was under-ame out the gate strong with some pretty impressive specs:

- Tegra 4 chipset
- 2 GB RAM
- 16 GB of internal storage
- 2×2 MIMO dual-channel Wi-Fi
- Bluetooth 4.0
- Ethernet port
- HDMI port
- Android L developer Preview

opers to
come up with anything they can to help this platform. It's far too early to tell what
elopers to
explore and develop for this device.

Exploring Android consoles

A small piece of adapter connected to a television and a controller device to control
budget
ease.

One of the first Android-based consoles is OUYA (image source: http://cdn2.pu.nl/media/misc/ouya_wall_ins.jpg):



These consoles are called microconsoles. A few years ago, the specification of such consoles was the following:

- **Model:** OUYA
- **Processor:** ARM Cortex A9
- **Speed:** 1.7 GHz quadcore
- **System chip:** NVIDIA Tegra 3
- **Flash memory:** 8 GB
- **RAM:** 1 GB DDR3
- **Display:** HD (720p) or Full HD (1080p)
- **Graphics processor:** NVIDIA GeForce ULP GPU
- **Android version:** Android 4.1 Jelly Bean
- **Connectivity:** Wi-Fi, Bluetooth and LAN

Now let's have a look at the modern age Android console specification:

- **Model:** NVIDIA Shield
- **Processor:** ARM Cortex A57 + A53 (64 bit)
- **Speed:** 1.9 GHz quadcore + 1000 MHz quadcore
- **System chip:** NVIDIA Tegra X1
- **Flash memory:** 500 GB HDD
- **RAM:** 3 GB
- **Connectivity:** Wi-Fi, Bluetooth, LAN, USB, and HDMI
- **Display:** 4K resolution support

Here is the NVIDIA Shield Android console (https://cdn0.vox-cdn.com/thumbor/72vPz7fqWT7ButeiG17cW_jjP2Y=/0x0:1920x1080/1600x900/cdn0.vox-cdn.com/uploads/chorus_image/image/45812214/shield-hero-image.0.0.jpg)



Let's now take a look at another modern age Android gaming console called Razor Forge TV (image source: <http://android.hu/img/2015/04/gallery-04.jpg>):



Its specifications are as follows:

- **Model:** Razor Forge TV
- **Processor:** Qualcomm Snapdragon 805
- **Speed:** 2.5 GHz quadcore
- **GPU:** Adreno 420
- **Flash memory:** 16 GB
- **RAM:** 2 GB
- **Connectivity:** Wi-Fi, Bluetooth, LAN, USB, and HDMI
- **Android version:** 5.0 Lollipop

A console is a specific device for gaming. However, nowadays, consoles can be used for various purposes, but the main objective remains the same.

From the previous example specifications, we can have an idea of how Android console gaming is improving. Developers work on a specific target device for

In a recent market study, it was said that PlayStation 4, Xbox One, and the Nintendo Wii U will be the dominant platforms for hardcore console gamers. However, Android console offerings from Amazon, Google, and others are projected to grow at a much faster rate, and offer the casual to mid-core gamer an affordable way to play of console gamers will likely consider Android.

This opens up a new era of Android game development. Console games are different to typical mobile games, which maximum Android game developers are into. However, with the growing number of Android consoles since 2014, more and more developers are taking interest in this.

Apart from the consoles discussed earlier, there are few more, such as the following:

- **Game Stick:** This is a small dongle-sized console powered by Android Jelly Bean, having 1 GB DDR3 RAM and 8 GB Flash memory. However, this specification is being boosted (image source: http://cdn2.knowyourmobile.com/sites/knowyourmobilecom/files/styles/gallery_wide/public/5/05/gamestick-4.jpg?itok=kYGDnKgr):



- **Mad Catz MOJO:** This is a normal microconsole having 2 GB RAM, 16 GB Flash memory, and a Tegra 4 processor. This device runs on Android 4.2.2 (image source: <http://cc.cnetcontent.com/inlinecontent/mediaserver/3m/8f8/607/8f8607ed9d2b4cee981f651125046895/original.jpg>):



- **GamePop:** BlueStacks has manufactured this next generation Android consoles. Most of the Android gaming consoles use store-based content. This is certainly a new venture with great expectations.
of
lass.
It would be wise to wait until the unit is in the wild before getting too excited unding.



The evolution of consoles running Android may be the future of gaming; however, it is established that the existence of other top consoles will not become extinct. The user base is increasing day by day, and so is the number of games on those platforms.

Being an Android game developer, one must not stick to the conventional gaming platforms such as smartphones and tablets. The era is changing rapidly. Developers should keep themselves up to date.

Exploring Android watches

et would

d from

very

flexible open source operating system, Android is one of the most favorite option for smartwatches (image source: <http://photos.appleinsidercdn.com/bigger-wimm-130830.jpg>):



ng games,
thus entering the game development domain. Advanced devices are as good as small computers. They consist of the Internet, sensors, cameras, Bluetooth, Wi-Fi, speakers, card slots, and have many more features.

or
ts or
nd GPS.
frontend for
or Wi-Fi.

Android Wear was first announced on March 18, 2014 by Google. At the same time, many manufacturers of electronic gadgets were announced as partners of Android Wear. These companies include Samsung, Motorola, LG, HTC, ASUS, and others.

lipop).

We can see a series of Android Watch releases around this period of time. LG started shipping LG G Watch, Motorola announced Moto 360, and ASUS released ZenWatch.

The latest advanced watches offer a set of attractive features. Users can find directions by voice from the phone, choose a transport mode, including a bike, and start a journey. While travelling, the watch shows directions, and will actually use tactile interaction to indicate turns by feel, helping the wearer travel without looking at the watch to play the Rolling Stones"). The screen then shows a card for play-control, volume, skip, and media images, and music can be controlled from the wrist with the user free to move around.

Let's look at a few specifications of Android wearable devices. First, the LG G Watch:

Processor	Qualcomm Snapdragon 400
Flash memory	4 GB
RAM	512 MB
Battery	400 mAh
Connectivity	Bluetooth 4.0
Sensors	Gyro, accelerometer, compass
Android version	4.3
Display	1.6 inch
Resolution	280x280

This configuration does not exist anymore in the market, but users do. So while count these configurations as well.

Android wearable manufacturers are also upgrading their devices with massive paring two releases:

	Sony Smartwatch 2	Sony Smartwatch 3
Processor	ARM Cortex M4	ARM A7 quadcore
Speed	180 MHz	1.2 GHz
Flash memory	256 MB	4 GB
RAM	64 MB	512 MB
Battery	225 mAh	420 mAh
Connectivity	Bluetooth 3.0	Bluetooth 4.0 and Wi-Fi ready
Sensors	Gyro, accelerometer, compass, proximity, ambient light, and IP57 dust and water resistant	Gyro, accelerometer, compass, proximity, ambient light, and IP68 dust and water resistant
Android version	4.0	4.3+
Display	1.6 inch	1.6 inch
Resolution	220x176	320x320

on here.

This is how the market is growing, and the applications as well.

Development insights on Android mobiles

As we discussed earlier, the main development target for any game developer on the Android platform are Android mobiles. We have also noticed the various technical specifications for Android mobiles. When a game developer targets this platform at

Mostly, all Android devices support a common touch interface, a physical Home button, a physical Lock button, a Back button, and volume Up-Down keys for user interaction. Besides these, an accelerometer can be also a good medium for the user interface.

Gaming is mostly visual, so game developers should always look for the graphic performance of the device. There is a separate graphics processor in the latest Android mobiles, but the quality varies.

ity also

e first-

in the LDPI

excellent, no

matter how hard the developer tried. Fortunately, with time, device manufacturers put in a lot of effort in to improving the visual quality along with performance. But this feature came with a price of battery life. The more quality it gained, the more battery it consumed.

Previously, few Android phones (for example, Android HTC Dream G1) had a physical QWERTY keypad. This made it much easier to port the game control system from Symbian or BlackBerry to Android (image source: <http://s.androidinsider.ru/2015/02/htc-dream.@750.jpg>):



Nowadays, the control system for Android games has changed completely to cope with the control style of other smartphone games in the market.

und 2750

display

quality. It is not possible to increase battery capacity beyond a certain limit due to the physical size and weight constraints for a mobile device.

Targeting the maximum devices is always a good idea as long as the balance between performance and gaming experience is maintained.

The device market is open for various devices. Although old configurations are is why developers have a minimum requirement for their games.

The

aster.

There was a time when BlackBerry was considered to be the only smartphone. elopers.

For Android mobile game development, a developer should keep in mind the following constraints and features:

- Small display area
- Wide range of resolution and pixel density
- Full-screen multitouch interface
- Sensor support for gyro, accelerometer, compass, ambient light, and so on
- Wide range of RAM
- A variety of processors and performance
- Battery life
- More chances of interruption

Android mobiles are one of the more profitable platforms now. When it comes id. So,

developers always jump into Android. Few economical reasons for the success of this platform are as follows:

- Availability of a massive user base, which attracts advertisers as an advertising platform
- Easy monetary transactions through well-established stores
- Ease of cross promotion of games and apps

Development insights on Android tablets

d
tablet. Tablet
ing reasons:

- **Bigger screen:** Although a bigger screen with the same resolution
sibility
le for a
small screen smartphone.
- **Bigger physical size:** Bigger physical size forces the player to play with both
hands, which results in better grip on the device and better controls.
- **Bigger space/playable area:** Bigger playable area can provide more control
e
touch control system, so he/she can concentrate more on the game alone.
Thus, it enhances the gaming experience.
- **Less constraints:** Continuous playing of games causes a serious amount of
nnected
blet
e of the
to
save power for jobs other than gaming.
- **Less interruption:** We all know how any interruption can be irritating during
an ongoing job. The same goes for the gaming experience as well. Any
rs
quit at that point in time. On a tablet, there are fewer chances of automated
interruption than manual or physical interruption; this means less irritation
while playing on a tablet.

ng
constraints and features in mind:

- Big display area
- Wide range of resolution, and comparatively low pixel density
- Full-screen multitouch interface
- Sensor support for gyro, accelerometer, compass, ambient light, and so on
- Wide range of RAM
- A wide range of processors and performance
- Fewer chances of interruption

Tablets evolved from the idea of a small portable computing device, which could be the bridge between smartphones and PCs/laptops.

A bigger screen always helps the user to interact with the game more easily. Game
he headache
me hardware
limitation.

Previously, most tablets used mobile processors, but tablet manufacturers are using laptop processors for tablets now. Intel Atom is an example of this. The more capable the processor used in a device, the better the quality it can deliver.

There was a time when Android games were targeted for mobiles first, and then those were ported for tablets. But the table has turned now. Now there is a very thin
Most of
he same

Development insights on Android TV and STBs

Firstly, Android TV game development requires a focus on two specific things:

- Large shared display
- Landscape resolution with lower dpi

increases the
overhead for graphic designers to optimize the assets accordingly. At the same time,
ake sure that
all the user actions can always be synced with the display.

TVs are big in comparison to any other Android devices. The stretch from a 5" to 50"
ints need to
be considered while developing games for Android TVs:

- Check the textures of the game—low resolution textures often look poor when stretched on Android TV
- 3D models might have jagged curves on TV because there are too few polygons
- Particle effects may need reworking for the TV's big screen if there are too few emitters, patterns, or colors
- Anti-aliasing is often not required on Android devices with small screens that have a high pixel density, but it effects a considerable visual difference for a TV.

e controllers

for the game. The TV can be directly controlled by some other Android device, or by a remote control.

However, any Android console or STBs can be used as well. In this case, a game controller or a D-pad control is much more useful for games.

To use a controller or a D-pad, a game developer should be very specific about using the proper control button for each functionality. When multiple players ap each player-controller pair.

It is an optional advantage to specify the game inside the `AndroidManifest.xml` file under the application tag, as follows:

```
<application
<!-- other declarations and tags -->
  android:isGame="true"
<!-- other declarations and tags -->
>
```

ll show it

under the games category on the Android TV home page.

uirements.

To declare support for game controllers, use the following code:

```
< uses-feature android:name="android.hardware.gamepad"
  android:required="true"/>
```

The developer must include the "touchscreen required false" declaration in the `AndroidManifest.xml` file, as it is used by Play Store for filtering. If it is missing, Play Store does not show the app to Android TV users in the search results:

```
< uses-feature android:name="android.hardware.touchscreen"
  android:required="false"/>
```

It is very good practice for all Android TV game developers to specify the non-example, a TV

does not have an accelerometer or gravity sensor, so marking them as non-required is a good development practice. This can be done as follows:

```
<manifest ...>
  <application ...>
    ...
    <!-- Requiring the camera removes this listing from Android TV
      search results -->
```

```

<uses-feature android:name="android.hardware.camera"
    android:required="true" />

<!-- Making accelerometer optional has no impact on Android TV
    filtering -->
<uses-feature android:name=
    "android.hardware.sensor.accelerometer"
    android:required="false" />
</application>
</manifest>

```

UI and game design

me, that is,
the screen

has to be designed accordingly. This is a major difference between mobile/tablet game development and Android TV game development.

As we are discussing the development on Android TV, a developer must consider how it would feel to play a game on Android TV from a distance of 4-10 feet while oper should look for:

- All text should be clearly readable
- UI buttons and other elements should match the overall layout without harming the readability and visibility of the whole screen
- A controller must control all the possible tasks while playing the game, as nobody would like to get up frequently to control the TV

Overscan

Unlike phone and tablet screens, TVs can lose some space at the edges of the screen to **overscan**. Although many TVs now use fixed-pixel technologies like LCD, many ide of the TV

screen free from important UI and gameplay elements. A good rule-of-thumb is drawing important elements.

If developers are using standard Android components for their UI, then they can use ean. If the

UI is custom OpenGL or OpenGL ES code, or is using a game engine's UI system, the developer will have to cater for overscan in the Android TV interface design.

Development insights on Android consoles

lity

game with limited processing power, comparable with PC and consoles. It is being les and

PC by the year 2015-2016. Now the question arises, "Will the console game market survive?" The answer is *Yes*.

Consoles are specially designed and configured to provide the best gaming

for

consoles, has proved to be a success.

There is not much difference between the configurations of an Android mobile and an Android console. Processor, memory capacity, and another few changes can be m. Mobiles,

typical

gaming controller.

Android gaming consoles are placed between mobiles and PCs from a development ection of

console game development depends on mainly two parameters: controls and the use of hardware.

We have already discussed the various types of consoles available in the market.

As a development platform, it does not have much uniqueness.

Development insights on Android watches

Wearable games run directly on the wearable device, giving the developer access to low-level hardware such as sensors, activities, services, and more, right on the so

required when the developer wants to publish to the Google Play Store. Wearables dheld

handheld

game is also useful for doing heavy processing, network actions, or other work and sending the results to the wearable.

To develop games on Android wearables, there are some technical steps to be followed. This is not general Android game development:

- The Android SDK tool has to be updated to version 23.0 or higher
- The Android platform support within the SDK has to be updated with Android 4.4.2 (API 20) or higher
- An Android wearable device or emulator is required for development

Creating and setting up a wearable application

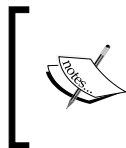
This can be done as follows:

Select **New Project** in Android Studio. In the **Configure Project** window, enter a name for the application and a package name. In the **Form Factors** window, perform the following steps:

1. Select **Phone and Tablet** and select **API 9: Android 2.3 (Gingerbread)** under **Minimum SDK**.
2. Select **Wear**, and select **API 20: Android 4.4 (KitKat Wear)** under **Minimum SDK**.
3. In the first **Add an Activity** window, add a blank activity for `mobile`.
4. In the second **Add an Activity** window, add a blank activity for `wear`.

When the wizard completes, Android Studio creates a new project with two modules, `mobile` and `wear`. Developers now have a project for both their handheld

stom layouts.
The handheld app does most of the heavy lifting, such as network communications,
tion. When
earable of the
results through notifications or by syncing and sending data to the wearable.



The `wear` module also contains a `Hello World` activity that uses a `WatchViewStub`. This class inflates a layout based on whether the device's screen is round or square. The `WatchViewStub` class is one of the UI widgets that the wearable support library provides.

Including the correct libraries in the project

There is a lot of library support for Android apps/games. Every developer needs to identify their correct requirements to include the correct libraries.

n wearable
devices:

- **Notifications:** The Android v4 support library (or v13, which includes v4) contains the APIs to extend the existing notifications on handhelds to support wearables. For notifications that appear only on the wearable (meaning, they are issued by an app that runs on the wearable), the developer can just use the standard framework APIs (API level 20) on the wearable, and remove the support library dependency in the `mobile` module of the game or application.
- **Wearable Data Layer:** To sync and send data between wearables and version of Google Play services. If developers are not using these APIs, remove the dependency from both modules.
- **Wearable UI support library:** This is an unofficial library that includes UI widgets designed for wearables. The Android platform encourages developers to use them in applications, because they exemplify best practices, and yet they can change at any time. However, if the libraries are updated, To get link only applicable if a developer creates wearable apps.

Hardware compatibility issues with Android versions

s, as we already know that Android is not compatible with ARM v4 processors, and Android 4.0+ requires ARM v7 or higher. Android wearables run on Android 4.4 or higher. So the developer must support ARM v7 onwards.

Platform-specific specialties

We have already discussed about all the Android hardware platforms till now. Each platform has its own specialties in terms of configuration, size, shape, utilities, and features.

Let's summarize the platform-specific points that should be taken into consideration while developing a game for the same.

Android mobiles

This type of Android hardware platform is the most famous and widely used device across the world. Typical Android mobile-specific features are:

- Small screen
- High dpi display
- Wide range of hardware configurations
- Full touchscreen
- Maximum sensor support
- Multipurpose use
- Maximum user base

Android tablets

This type of Android hardware platform is the second most famous and widely used devices across the world, with slightly different utilities. Typical Android tablet-specific features are:

- Comparatively bigger screen
- Low dpi display
- Full touchscreen
- Specific use device

Android televisions and STBs

with
pical Android
television- and STB-specific features are:

- Biggest display unit
- No touch interface
- D-pad or controller-based input system
- Fixed landscape orientation
- Limited hardware support
- Suitable for multiplayer games

Android consoles

Beside famous gaming consoles such as the PS3, PS4, and Xbox, Android gaming consoles are also gaining popularity nowadays. Typical Android console-specific features are:

- Dedicated hardware system for gaming
- Full controller-based input system
- Multiresolution large display support
- Hardware-specific development
- Best Android platform for multiplayer gaming experience

Android watches

this device

this device.

Typical Android watch-specific features are:

- Very small display
- Limited hardware support
- Less memory and processing power
- Touchscreen interface
- Very portable
- Separate wearable development environment needed

Summary

hardware and software specifications. In this chapter, you have learned about the possible different hardware platforms running on Android. Through this knowledge, it is easier to

choose a specific set of hardware platforms to target.

In the near future, Android will be stepping into the world of virtual reality with wearables. Now, you have

come to know that mobiles, tablets, televisions, STBs, consoles, and watches are the various hardware platforms. All of them are capable of running Android games. However, Android consoles are the only dedicated hardware platform for games.

Though consoles are a dedicated gaming platform, Android mobiles and Android
I the
platforms
these
platforms to acquire as many users as possible.

3

Different Android Development Tools

We have already discussed the different Android target devices for game and tools, it is very important to know about the helpful software that can make the development process easier and effective.

Android game development is supported or backed by many powerful tools and development process:

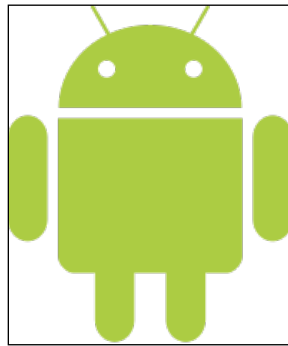
- Android SDK
- Android Development Tool
- Android Virtual Device
- Android Debug Bridge
- Dalvik Debug Monitor Server

for the Android platform. Although ADB and AVD are not mandatory for development, virtual devices in order to debug the game.

Android SDK

Android SDK is the main development kit required to build any application for skeleton for any Android development. This SDK itself comes with dozens of support tools. It contains platform details, APIs, and libraries along with ADT and AVD. So having necessary st platforms and other tools.

selection is manual, and it is recommended to have only the necessary platforms as latform along with the minimum targeted version of Android (image source: http://photos4.meetupstatic.com/photos/event/1/1/0/f/highres_441724367.jpeg):



Android Development Tool

Android Development Tool (ADT) is a plugin for the Eclipse IDE that is designed to ations.

up new
Android projects, create an application UI, add packages based on the Android
ven
export signed (or unsigned) .apk files in order to distribute the application.

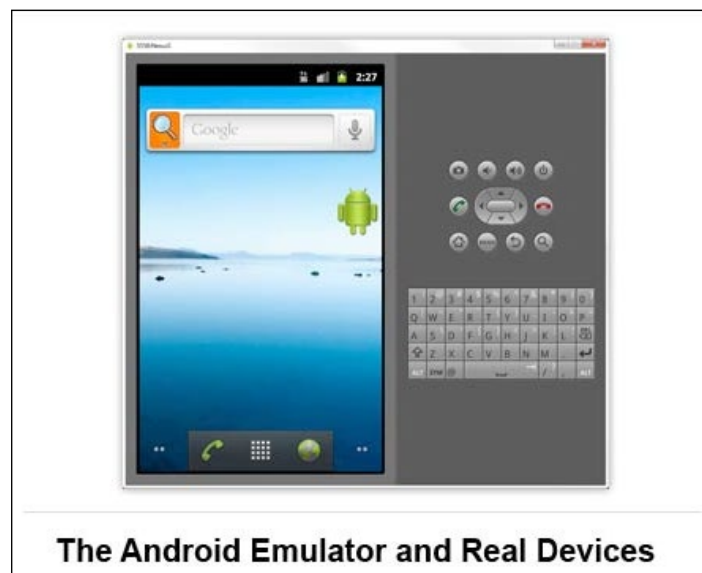
Developing in Eclipse with ADT is highly recommended, and is the fastest way to integration,
custom XML editors, and the debug output pane, ADT gives an incredible boost to developing Android applications.

However, ADT support for Eclipse is being pulled by Google, so developers are recommended to switch to Android Studio.

Android Virtual Device

An **Android Virtual Device (AVD)** is a software-produced model of a real device, which can be configured with custom hardware specifications. It can be a virtual copy of the real device as well. This is one of the most important tools for any Android developer. This lets the developer test the application in a typical Android development

time (image source: <http://www.geeknaut.com/images/2014/08/top-android-emulators-for-windows3.png>):



Configuring AVD

An AVD consists of the following:

- **Hardware profile:** This profile describes the hardware features of the virtual device. This can be configured with hardware options like a QWERTY keypad, camera, integrated memory, and so on.
- **System image mapping:** The running Android platform version can be configured depending on the installed set of Android platforms. Android platforms can be installed by the Android SDK manager.

- **Dedicated disk space:** Dedicated storage area on your development machine or a virtual SD card.
- **Other features:** The developer can even specify the look and feel of the virtual device such as device skin, screen dimension, and appearance.

er, which
is located in the <SDK Path>/tools directory:

1. On the main screen, click on **Create Virtual Device**.
2. In the **Select Hardware** window, select a device configuration such as **Nexus 5**, then click on **Next**, then click on **Finish**.
3. To begin customizing the device by using an existing device profile as a template, select a device profile and then click on **Clone Device**. Or, to create a complete custom emulator, click on **New Hardware Profile**.
4. Set the following to create a new custom emulator:
 - Device name
 - Screen size
 - Screen resolution
 - RAM
 - Input options
 - Supported states
 - Camera options
 - Sensor options
5. After setting every property, click on **Finish**.

The developer can also create a new custom emulator using the command line, as follows:

```
android create avd -n <name> -t <targetID> [-<option> <value>] ...
```

Here, the following options can be set:

- name: This will be the custom AVD name
- targetID: This will be the custom ID
- option: This can include options such as device screen density, resolution, camera, and so on.

The developer can execute this command to use a previously defined AVD:

```
android list targets
```

Then, the developer can run the following command:

```
emulator -avd <avd_name> [options]
```

Android Debug Bridge

Android Debug Bridge (adb) is a tool used to establish communication between the development environment and a virtual device or the connected Android device. It is a client-server command-line program, which works on three elements:

- **Client on the development machine:** Works as the client, which can be invoked by adb commands. Other Android tools such as the ADT plugin and DDMS also create adb clients.
- **Daemon:** A background process that runs in the background on each emulator or device instance.
- **Server on the development machine:** This is a background process that runs on the development machine and manages the communication between the client and server.

On starting adb, the client first checks whether there is an adb server process already tarts, it binds to the local TCP port 5037 and listens for commands sent from adb clients – all adb clients use port 5037 to communicate with the adb server.

nces. It

locates emulator/device instances by scanning odd-numbered ports in the range 5555 to 5585, the range used by emulators/devices. Where the server finds an adb vice instance connections and an odd-numbered port for adb connections.

veloper

can use adb commands to access those instances. Because the server manages connections to emulator/device instances, and handles commands from multiple any client (or from a script).

Using adb on an Android device

One of the first things to remember is to put the development device in the USB debugging mode. This can be done by navigating to Settings, tapping on **Developer options**, and checking the box named **USB debugging** for Android 5.0 and above (for other Android versions, refer to <https://www.recovery-android.com/enable-usb-debugging-on-android.html>). Without doing this, the development PC won't recognize the device.

via the command line. This is done with the `cd` (change directory) command. So, if (on Windows) the SDK folder is called `android-SDK`, and it's in the root (`c:`) directory, you can enter the following command:

```
cd c:/android-SDK
```

Then, to get into the adb folder, use this:

```
cd platform-tools
```

At this point, the prompt will say this:

```
C:\android-SDK\platform-tools>
```

After locating and installing the drivers for a particular device:

```
adb devices
```

checked. The phone doesn't say "Droid Razr" or "Galaxy Nexus".

is a task in couldn't device, prepared in advance.

Beside the specific instructions to root a particular device, the next thing the developer needs will be the drivers for the phone or tablet.

The easiest way to do this is usually to simply Google search for the *specific device plus drivers*. So if the developer has a Droid Razr, he/she should search for `Droid Razr Windows Drivers`. This will almost always direct the developer to the best link.

Another option, which will only work for stock Android devices, is to download the USB drivers from the SDK. To do this, launch the SDK manager again. Go to the **Available packages** tab on the left, expand the **Third party add-ons** entry, and then expand the **Google Inc. add-ons** entry. Finally, check the entry for the **Google USB Driver** package.

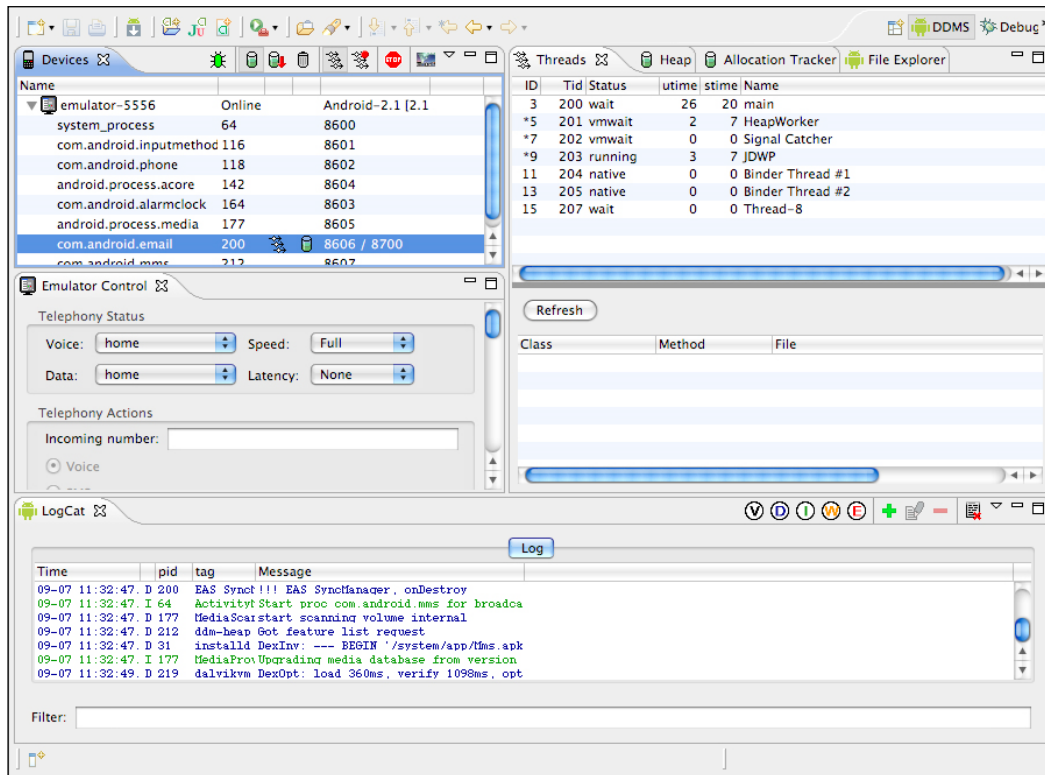
Note that the USB driver package isn't compatible with OS X.

Dalvik Debug Monitor Server

The **Dalvik Debug Monitor Server (DDMS)**, whether it's accessed through the `ovides`

handy features for inspecting, debugging, and interacting with emulator and device instances. You can use DDMS to inspect running processes and threads, explore the `fid` even

take screenshots. For emulators, you can also simulate mock location data, send SMS messages, and initiate incoming phone calls:



As the preceding screenshot shows, DDMS can primarily track, update, and display the following information:

- All running processes
- All running threads per process
- Consumed heap per process
- All log messages

On Android, every application runs in its own process, each of which runs in its VM.

DDMS connects to adb on start. On successful connection, a VM monitoring service is created between adb and DDMS, which informs DDMS upon starting and ending a VM on the device. DDMS retrieves the VM's process ID via adb, and opens a connection to the VM's debugger when there is an active VM running through the adb daemon on the device. DDMS can now communicate to the VM using a custom wire protocol.

DDMS also listens on the default debugging port, called **base port**. The base port is a port forwarder, which can accept VM traffic from any debugging port and forward it to the debugger. The traffic that is forwarded is determined by the currently selected process in the DDMS **Devices** view.

Other tools

The elements mentioned in the previous sections are the minimum requirement for Android development, with which a full application can be created. However, the development process can become much easier with the support of a few other tools. Let's have a look at a few of such tools. These tools are not mandatory for Android development, but they are recommended to be used for a better development process.

Eclipse

is used
partially due
to tools with

Eclipse. This integration is achieved with the ADT plugin for Eclipse, which can be downloaded from the Android website.

Use of Eclipse for Android development is a well-known practice for many developers. Some of the reasons for this are as follows:

- Free Eclipse IDE
- Direct Android plugin support
- Direct DDMS support
- Simple interface
- Android NDK support

The launch of Android Studio reduced the popularity of Eclipse among Android developers, because Android Studio has everything inbuilt to support any Android development. Moreover, it is a much simplified tool to use in design view. Google itself is promoting the new tool massively.

There are a few drawbacks in Eclipse Android development, because it uses Android SDK as a third-party tool. The significant drawbacks are as follows:

- Debugging through Eclipse is sometimes difficult
- ADB configuration is tricky
- Android manifest has to be managed manually
- The design view is very complex through Eclipse IDE

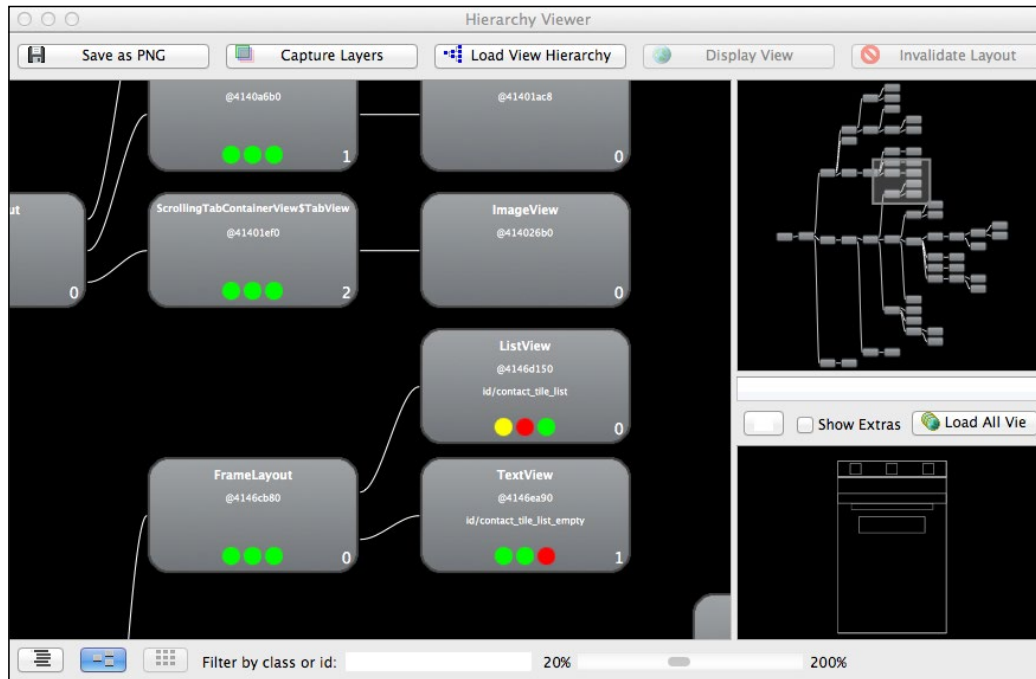
Eclipse is an excellent standalone IDE, but when it comes to Android development, Android Studio wins the race.

Hierarchy Viewer

on or the relatively new Eclipse perspective, is used to see how your layouts and screens resolve at runtime.

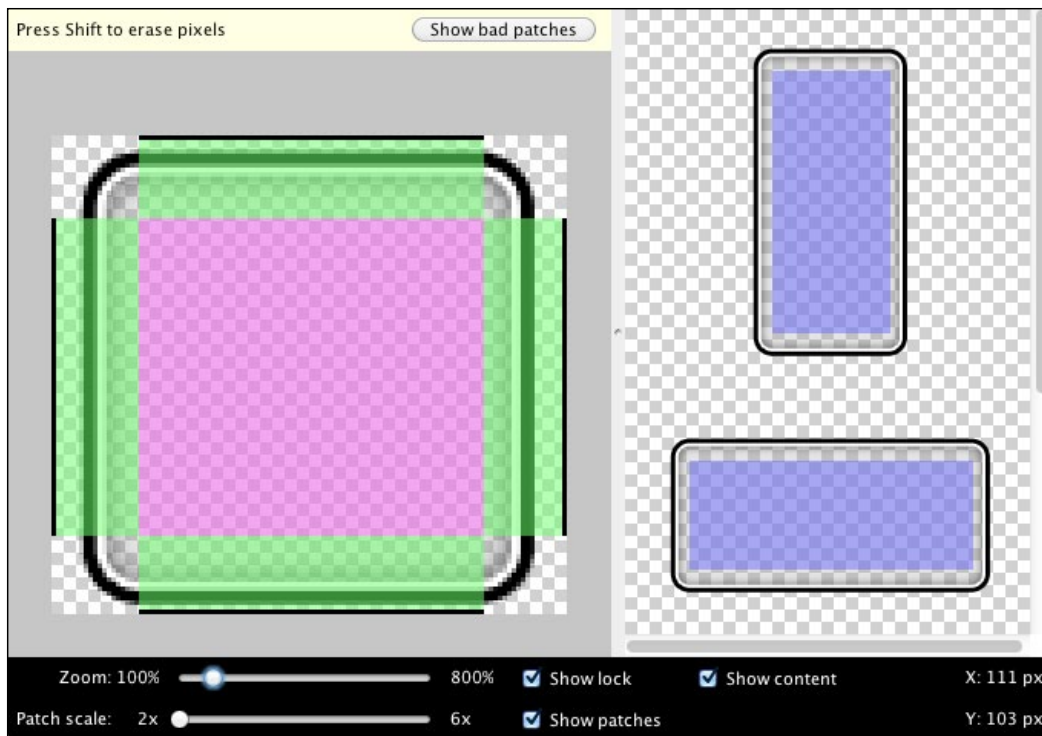
of your

https://media-meditemple.netdna-ssl.com/wp-content/uploads/2012/03/da_hierarchy_viewer.png);



Draw 9-Patch

When it comes to graphics design, the Draw 9-patch tool comes in handy. This tool allows you to convert traditional PNG graphic files into stretchable graphics that are more flexible and efficient for mobile development use. The tool simplifies the creation of NinePatch files in an environment that instantly displays the results:



ProGuard

ProGuard is not directly associated with Android development, but it helps in protecting the intellectual property of the developer. It is a very common practice for Android game developers to use ProGuard.

structure. This tool can be configured to obfuscate the resulting binary. ProGuard also helps in optimizing the binary, so that the overall package size is reduced.

ProGuard can be difficult to use when developers try to integrate pre-compiled JARs into the Android project. Sometimes conflicts are created in class structures if the must be configured to exclude the precompiled JARs in order to achieve a successful build.

It is always recommended to use ProGuard to protect the game classes from reverse engineering or decompilation.

Asset optimization tools

We all know the vast range of Android hardware configurations. It is always necessary to optimize the assets to reduce runtime memory usage and unnecessary data processing. In games, graphical assets take most of the storage and memory.

Full asset optimization

An unoptimized asset may hold some unnecessary data, like transparency, alpha bit depth, and so on.

It is risky to use such tools continuously while developing the game using these tools.

Optimizing assets with an 8-bit color palette is not a good option. Optimization is never recommended for any games, and it is the developer's responsibility to use proper optimization techniques to maintain the game quality.

The following are a few such asset optimizer tools:

- PNGOUT
- TinyPNG
- RIOT
- JPEGmini
- PNGGauntlet

Reducing the size of the assets is one of the following reasons:

- The developers do not optimize each asset separately, which results in quality loss for a few assets.
- It is really difficult to choose the right tools for optimization. Sometimes, multiple tools are required for the same job, which slows down the overall development process.

Creating sprites

In many cases, it is noticed that a large number of small art assets are being used in games individually. This may cause critical performance lag for the game. It is recommended to use a sprite building tool to merge those assets into one to save space and time. SpriteBuilder and TexturePacker are two good examples of such tools.

Tools for testing

For any development process, testing is of major importance. For Android game development too, there are a few tools and processes to make testing easier.

Creating a test case

Activity tests are written in a structured way. Make sure to put your tests in a test package name should follow the same name as the application package, suffixed with `.tests`. By convention, your test case name should also follow the same name as the Java or Android class that you want to test, but suffixed with `Test`.

To create a new test case in Eclipse, perform the following steps:

1. In Package Explorer, right-click on the `/src` directory for your test project, and select **New | Package**.
2. Set the **Name** field to `<package_name>.tests` (for example, `com.example.android.testingfun.tests`), and click on **Finish**.
3. Right-click on the test package you created, and select **New | Class**.
4. Set the **Name** field to `<activity_name>Test` (for example, `MyFirstTestActivityTest`), and click on **Finish**.

Setting up your test fixture

A test fixture consists of objects that must be initialized for running one or more tests. To set up the test fixture, you can override the `setUp()` and `tearDown()` methods in your test. The test runner automatically runs `setUp()` before running any other test methods, and `tearDown()` at the end of each test method execution. You can use rate from the tests methods.

To set up a test fixture in Eclipse, follow the steps listed next:

1. In Package Explorer, double-click on the test case that you created earlier to tend one of the subclasses of `ActivityTestCase`. For example:

```
public class MyFirstTestActivityTest extends ActivityInstrumentationTestCase2<MyFirstTestActivity> {
```

2. Next, add the constructor and `setUp()` methods to your test case, and add variable declarations for the activity that you want to test. For example:

```
public class MyFirstTestActivityTest
    extends ActivityInstrumentationTestCase2<MyFirstTestActivity> {

    private MyFirstTestActivity mFirstTestActivity;
    private TextView mFirstTestText;

    public MyFirstTestActivityTest() {
        super(MyFirstTestActivity.class);
    }

    @Override
    protected void setUp() throws Exception {
        super.setUp();
        mFirstTestActivity = getActivity();
        mFirstTestText =
            (TextView) mFirstTestActivity
                .findViewById(R.id.my_first_test_text_view);
    }
}
```

The constructor is invoked by the test runner to instantiate the test class, while the `setUp()` method is invoked by the test runner before it runs any tests in the test class.

Typically, in the `setUp()` method, you should invoke the superclass constructor for `setUp()`, which is required by JUnit

You can initialize your test fixture state by:

1. Defining the instance variables that store the state of the fixture.
2. Creating and storing a reference to an instance of the activity under test.
3. Obtaining a reference to any UI components in the activity that you want to test.

Developers can use the `getActivity()` method to get a reference to the activity under test.

Adding test preconditions

As a sanity check, it is good practice to verify that the test fixture has been set up correctly, and the objects that you want to test have been correctly instantiated or

hing was wrong with the setup of your test fixture. By convention, the method for verifying your test fixture is called `testPreconditions()`.

For example, you might want to add a `testPreconditions()` method like this to your test case:

```
public void testPreconditions() {
    assertNotNull("mFirstTestActivity is null",
        mFirstTestActivity);
    assertNotNull("mFirstTestText is null", mFirstTestText);
}
```

The assertion methods are from the Junit Assert class. Generally, you can use assertions to verify if a specific condition that you want to test is true.

If the condition is false, the assertion method throws an `AssertionFailedError` exception, which is then typically reported by the test runner. You can provide a string in the first argument of your assertion method to give some contextual details if the assertion fails.

r proceeds to run the other test methods in the test case.

Adding test methods to verify an activity

ehavior of your activity.

thod like this to check that it has the correct label text:

```
public void testMyFirstTestTextView_labelText() {
    final String expected =
        mFirstTestActivity.getString(R.string.my_first_test);
    final String actual = mFirstTestText.getText().toString();
    assertEquals(expected, actual);
}
```

The `testMyFirstTestTextView_labelText()` method simply checks that the default text of the `TextView`, which is set by the layout, is the same as the expected text defined in the `strings.xml` resource.



When naming test methods, you can use an underscore to separate what is being tested from the specific case being tested. This style makes it easier to see exactly what cases are being tested.

read the expected string from your resources instead of hardcoding the string in your comparison code. This prevents your test from easily breaking whenever the string definitions are modified in the resource file.

To perform the comparison, pass both the expected and actual strings as arguments to the `assertEquals()` method. If the values are not the same, the assertion will throw an `AssertionFailedError` exception.

If you added a `testPreconditions()` method, put your test methods after the `testPreconditions()` definition in your Java class.

pse. To build and run your test, follow these steps:

1. Connect an Android device to your machine. On the device or emulator, open the **Settings** menu, select **Developer options**, and make sure that **USB debugging** is enabled.
2. In the Project Explorer, right-click on the test class that you created earlier, and select **Run As | Android JUnit Test**.
3. In the **Android Device Chooser** dialog, select the device that you just connected, then click on **OK**.
4. In the JUnit view, verify that the test passes with no errors or failures.

Performance profiling tools

he CPU
y stores
e pieces of
your app to
be slow, worsens the display performance, or exhausts the battery.

To discover what causes your specific performance problems, you need to take a
n behavior,
see, and
improve your code.

Android Studio and your device provide profiling tools to record and visualize the
rendering, computing, memory, and battery performance of your app.

Android Studio

Android Studio is the official IDE for Android application development, based on
oid Studio
offers the following among many others:

- Flexible Gradle-based build system
- Build variants and multiple .apk file generation
- Code templates to help you build common app features
- Rich layout editor with support for drag and drop theme editing
- lint tools to catch performance, usability, version compatibility, and other problems
- ProGuard and app-signing capabilities
- Built-in support for the Google Cloud platform, making it easy to integrate Google Cloud messaging and App Engine

If you're new to Android Studio or the IntelliJ IDEA interface, this section provides an introduction to some key Android Studio features.

Android project view

By default, Android Studio displays your project files in the Android project view. This view shows a flattened version of your project's structure, which provides quick access to the key source files of Android projects, and helps you work with the Gradle-based build system. The Android project view:

- Shows the most important source directories at the top level of the module hierarchy
- Groups the build files for all modules in a common folder

- Groups all the manifest files for each module in a common folder
- Shows resource files from all Gradle source sets
- Groups resource files for different locales, orientations, and screen types in a single group per resource type

The Android project view shows all the build files at the top level of the project hierarchy under `Gradle Scripts`. Each project module appears as a folder at the top level:

- `java/`: Source files for the module
- `manifests/`: Manifest files for the module
- `res/`: Resource files for the module
- `Gradle Scripts/`: Gradle build and property files



For example, the Android project view groups all the instances of the `ic_launcher.png` resource for different screen densities under the same element.

The project structure on disk differs from this flattened representation.

the **Project** drop-down menu.

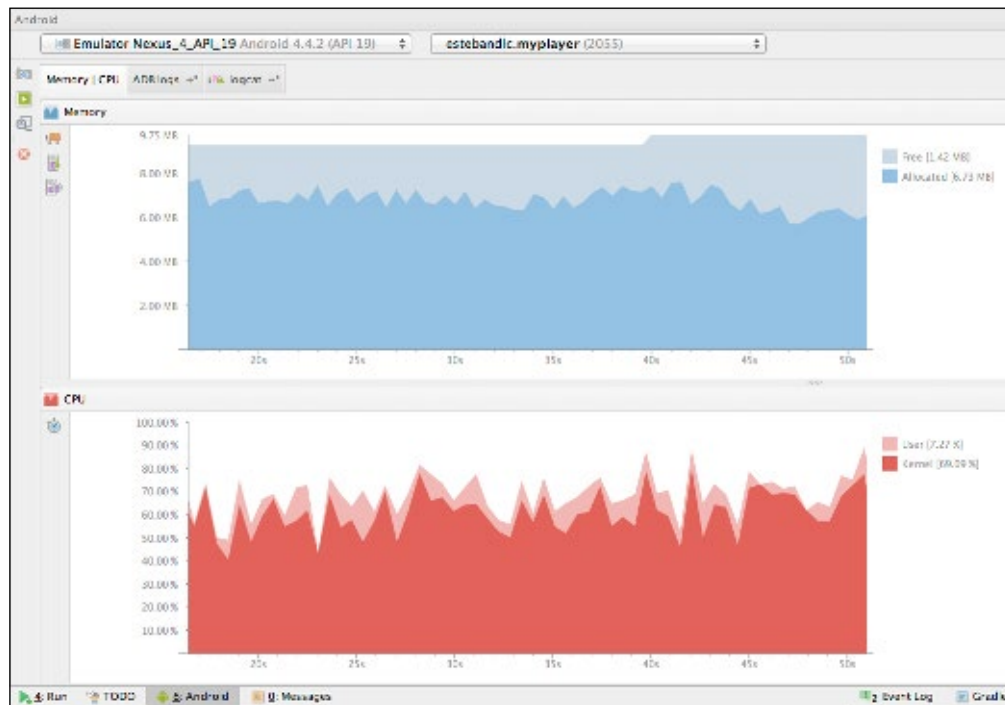
Memory and CPU monitor

Android Studio provides a memory and CPU monitor view so that you can easily monitor your app's performance and memory usage to track CPU usage, find deallocated objects, locate memory leaks, and track the amount of memory the

, click on the **Android** tab in the lower-left corner of the runtime window to launch the **Android runtime** window. Click on the **Memory | CPU** tab.

garbage collection, and dump the Java heap to a heap snapshot in an Android-specific HPROF binary format file at the same time. The HPROF viewer displays classes, instances of each class, and a reference tree to help you track memory usage and find memory leaks.

Android Studio allows you to track memory allocation as it monitors memory use. allocated to adjust ance and memory use.



Cross-platform tools

Although we are only talking about Android game development, game development cannot be efficient without cross-platform support. We have already discussed game design flow deploy the game for various platforms such as iOS, Windows, consoles, and the like.

s simple as

writing the code once, putting it through a tool for translation, and publishing both an iOS and Android app to the respective app stores.

Using a cross-platform mobile development tool can reduce the time and cost
e updated
o so that
the menu and control commands match the UX of how Android devices and iOS
devices inherently operate in different ways.

take a look
at a few of them:

Cocos2d-x

developers the option of five different forks or platforms to develop on, based on
e C,
Python, and C# (image source: http://www.cocos2d-x.org/attachments/802/cocos2dx_landscape.png):



Primarily, this tool is efficient for Android, iOS, and Windows Phone. The
development platform is mainly 2D; however, from Cocos2d-x 3.x it is possible to
develop games in 3D also.

Cocos2d-x works with native Android, and can support different processor
architectures separately. This tool works in a Unix-based environment.

There is a huge developer community that develops games on Cocos2d-x. Here are
the pros and cons of this cross-platform development engine from the Android game
development perspective:

The pros are as follows:

- Supports the most common programming languages such as C++
- Works in the native environment
- Lightweight and optimized library
- Common OpenGL rendering system
- All smartphone features supported for 2D development
- Completely free open source Engine

The cons are as follows:

- Majorly supports 2D development
- Cross-platform deployment is tricky and complicated
- Performance and memory optimization is weak
- No visual programming support
- No debugging tool is provided within the engine
- Mostly works on mobile phone platforms

Unity3D

game

powerful

multiple, PC,

Mac, consoles, web, Linux, Xbox, PlayStation, and so on. Currently, it supports 17 different platforms for game development (image source: http://img.danawa.com/images/descFiles/3/545/2544550_1_1390443907.png):



to help

you distribute it to the appropriate stores, get social shares, and track user analytics.

Almost

can find

an effective prebuilt custom library, prebuilt plugins, and so on, which helps a developer in terms of reducing development time significantly. Here are the main pros and cons of Unity3D.

The pros are as follows:

- Supports 17 different platforms for gaming
- Very simple deployment procedure
- Visual editor to support visual programming
- Inbuilt powerful debug tool
- Huge library support
- Hassle-free development
- Inbuilt powerful memory and performance optimizer

The cons are as follows:

- Comparatively bigger library size
- Slightly performance heavy (however, it is improving day by day)
- Supports only scripting languages (C#, JavaScript, and Boo)
- Not completely free for commercial purposes
- Mainly works well with 3D

Unreal Engine

game
only, but
and iOS

(image source: <http://up.11t.ir/view/691714/1425334231-unreal-engine-logo.png>):



There have been a lot of debates about whether Unreal Engine 4 is better than
the pros
and cons of Unreal Engine 4:

The pros are as follows:

- The Blueprint feature allows flexible visual programming
- Generic C++ language is more developer friendly
- Graphic processing is excellent
- Inbuilt dynamic shadow system
- Simple to understand and start making games
- Vast support in terms of device scalability
- In-editor material designing

The cons are as follows:

- Mobile optimization is still not up to the mark
- Lack of 2D development tools
- Lack of availability of third-party plugins
- Working with sprites is a pain for mobile development
- Still focused on high configuration hardware platforms

PhoneGap

Owned by Adobe, PhoneGap is a free resource that first-time app developers can use to translate code from HTML5, CSS, and JavaScript.

p an app
s completed,
make any
improvements.

Beyond iOS and Android, PhoneGap also creates apps for BlackBerry and Windows.

[http://
blogs.perceptionssystem.com/wp-content/uploads/2016/03/phonegap.png](http://blogs.perceptionssystem.com/wp-content/uploads/2016/03/phonegap.png)):



PhoneGap features the following pros:

- Supports almost all mobile platforms
- Lightweight application build
- Supports HTML, CSS, and JavaScript
- Cordova apps install just like a native application
- PhoneGap is open source and free

It has the following cons:

- Lack of platform support
- Lack of third-party plugins
- Native UI is still difficult to use

Corona

Corona's SDK comes with the promise that you can start coding your new app in as little as five minutes after the download. It's another cross-platform mobile make

(image

source: <https://qph.ec.quoracdn.net/main-qimg-fad64a16e531773325448e6ca699d117>):



ally

robust, with a small footprint for mobile apps.

Corona has the following pros:

- Good application performance in terms of FPS
- Good inbuilt emulator
- Light application build

It has the following cons:

- Uses the less popular scripting language Lua
- Not free
- Less plugin support
- No on-device debugging support

Titanium

Using JavaScript, Titanium's SDK creates native iOS and Android apps while reusing anywhere from 60% to 90% of the same code for all the apps you make, thereby saving you a significant amount of time (image source: <http://mobile.e20lab.info/wp-content/uploads/sites/2/2014/04/titanium.png>):



Because Titanium is an open-source tool, hundreds of thousands of your fellow developers are constantly contributing to it to make it better, and give it more functionality. And if you happen to find a bug in its system, you can do so too.

The pros are as follows:

- Quick-start flexibility for the initial phase
- Lightweight application build
- Common JavaScript language
- Web and mobile support on Android and iOS
- Open source

The cons are as follows:

- Lack of plugin support
- Lack of platform support range
- Script-based development increases complexity and effort
- Performance varies with different platforms
- Poor optimization compared to other tools

Summary

have
stage.

The necessity for these tools is realized when they are required.

We have discussed all the mandatory tools for Android development only. But modern age game development demands flexibility across hardware platforms as well as operating systems. This is where cross-platform development engines come in and more efficient a larger

build size. In most cases, developers have limited control over the cross-platform engine, but full control can be gained if the game is developed on native SDKs.

they are

very efficient in optimizing the game along with data protection, which might not have a direct impact on games. A good developer must use optimization tools to deliver a better performing game.

4

Android Development Style and Standards in the Industry

an the syntactical grammar. However, most developers across the globe follow a few fundamental styles and standards for writing Android code. Android is based on Java, so most of the stylization follows Java standards.

When it comes to Android game development, there are a few design styles that should be followed. They do not cover game design, rather more technical design. These kinds of styles and standards indicate a proper project structure, class structure, and folder/file management.

Typical game design also involves following some rules while working on the ms of game design.

gh the following topics:

- The Android programming structure
- Game programming specifications
- Technical design standards
- Game design standards
- Other style and standards
- Different styles for different development engines
- Industry best practices

The Android programming structure

Android style or recommendation is not a definite programming rule. However, a good programming practice always includes a set of rules. To code in Android, the code structure follows the Android base structure and hierarchy.

id
style.

Class formation

Java class formats should be consistent and follow the Javadoc rule; a standard structure should follow this sequence:

1. Copyright information
2. License information
3. Package declaration
4. Library imports
5. Class description and purpose
6. Class definition
7. Global variables
8. Constructor
9. Methods

This is the copyright and license information format:

```
/*
 * Copyright (C) <year> authority
 *
 * <License information and other details>
 */
```

This is the class and method description format:

```
/*
 * <Description>
 * <Purpose>
 */
```

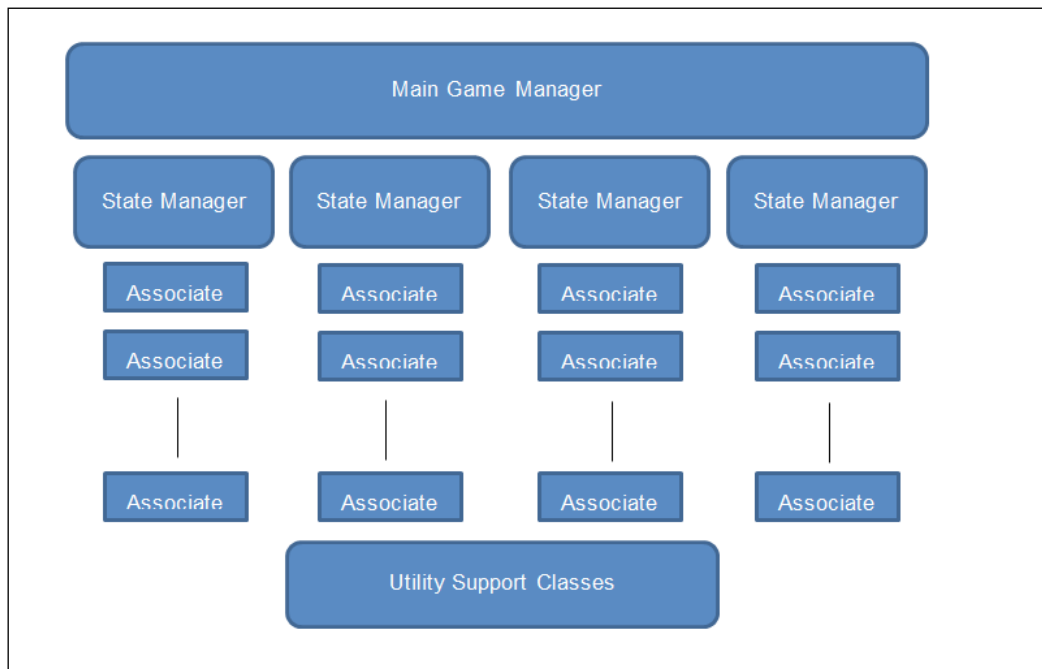
Call hierarchy

Like the coding style, there is no defined call hierarchy. However, in Android are mainly three kinds of classes in the project structure:

- Managers and controllers
- Associates
- Utility classes

A game requires the managers and controllers to implicate game rules and regulations. It is also used to control the behavior of game elements and states. Most games consist of multiple sections or screens, for example, menu, level selection, game play, and so on.

These sections can be termed as states, and the elements used inside these are called associates. Associates may be separate classes by themselves. Utility classes basically support development by providing predefined functionality, such as in-game sound implementation, record store management, common utilities, network connectivity, and so on:



Call hierarchy

Inside the game loop, the main Android game thread loops hand over control to on the e manager gers, depending out the game cycle and manages the main game activity as well.

election, in-gameplay, and so on) and all the required associate classes. State managers pass the call to the respective associates to carry out a specific task.

g on the requirements.

sign the call hierarchy differently. Everything is good, and the game flow is also running properly without sacrificing the class/data security.

Game programming specifications

er.

programming. The sense of game design is also different.

ion is meant e serves a very different purpose than a software or an application.

This is why game programming has to follow a few specifications and protocols.

Game programming can be divided into these following categories:

- Gameplay programming
- Graphics programming
- Technical programming

Gameplay programming

l. A sense of game design is also required. Gameplay programming includes AI, user control, physics, display control, and so on.

igners
for prototyping the game.

Graphics programming

are
manipulate
graphic assets while the game is running.

Graphics programming is all about getting the maximum from the GPU. Nowadays, most games are graphics heavy. The latest devices have separate, powerful GPUs to support heavy graphics.

l new 3D
on entirely
ual quality
without affecting the game performance.

Most of the time, game performance drops significantly due to heavy art asset processing, which is never a desired situation. Shaders or graphics programmers have to balance quality and performance.

This is mostly effective when games are ported across different platforms. As we have discussed earlier, a range of hardware platforms with a wide range of hardware specifications run on Android. Separate shaders and assets are required for this kind each pixel in a particular way.

Technical programming

res
n
ing.

Here is a list of the possible areas for technical programming:

- Sound programming
- Network programming
- Game tool programming
- Research and development programming

Sound programming

round

music. Sound programming has therefore become a part of game programming. A sound programmer mainly has knowledge of digital signal processing. Sound programmers have to work with sound designers.

Modern-age games use 3D sound systems. Sound programming plays a critical role in delivering quality sound without affecting the performance.

Network programming

time,

the game did not communicate with other instances, so there was not much need for network programming at that point in time. The modern age is the age of common

today; even standalone games communicate with other game instances just for socialization and monetization.

Network programming takes care of network latency, packet optimization, er

is also responsible for managing client-server communication and creating the architecture.

Some games run on the server. The client acts as a display device for the game g. Few

games even follow asynchronous communication. Network programming assures the smooth and proper transition in this architecture.

Game tool programming

s. Tool

oper game

tool programming can ease the development process a lot. A lot of time and effort can be saved with the help of development tools.

We have already discussed a few development tools. However, it is not necessary can be game specifi used in a game.

Research and development programming

is programming helps make gaming better and finds new techniques to be used for upcoming games.

ng, hardware platforms, and native development. Programmers should have knowledge of native language and assembly or hardware-level language as well.

In the case of Android game development, research programmers are assigned to explore new Android devices along with a new feature and specification. These ible way in games.

e use of various sensors such as gravity, light, accelerometer, and so on. The recent le of such experiments.

Technical design standards

Mostly, game development revolves around game design; however, the development process is controlled by technical design. Technical design considers l game design and requirements.

A technical design contains the following sections:

- Game analysis
- Design pattern and flow diagram
- Technical specification
- Tools and other requirements
- Resource analysis
- Testing requirements
- Scope analysis
- Risk analysis
- Change log

Game analysis

This section of technical design analyzes the game design thoroughly and figures out what the development is dependent on. Then this section is also considered in technical design.

Design development process. This helps define the timeline and predict upcoming challenges, with possible solutions.

Design pattern and flow diagram

This section designs the class diagram and hierarchy for the game. The game flowchart and server-client architecture (if required) are also defined here.

This section of game technical design gives a clear picture of upcoming development for a developer. Each and every part of game modules, program structure, call hierarchy, third-party tool integration, database connectivity, and server-call management should be clearly declared in this part of the technical design document.

The visual display of such a diagram, showing the flowcharts, is always a headstart for any development process.

Technical specification

The technical specification specifies the development platform, target device set, and software is required to develop the project.

This identification is essential before starting the actual development. For any supporting development with the hardware also identified in this section of technical design.

So, basically, there are two different sections in this specification. First is to specify and identify the system required to create the game according to the design.

Tools and other requirements

em

requirements. In many cases, this section is included in the technical specification. However, this part serves a different purpose.

This may create the requirement to develop a new tool for the actual game development. Therefore, tool programmers are referred to this section. Android game development is not out of scope for this section. Although most of the tools are readily available for Android development, a few scenarios may demand a game-specific tool as well. In this case, the tool design and separate technical design, and the use of the tool is mentioned in this section.

Resource analysis

chnical

dependency, and other resource dependencies. This helps estimate project cost and decide the development timeline.

Testing requirements

al part of a

game development process. Technical design should define the testing procedure along with the defined test cases.

The development head of the game identifies the stages of testing and its requirements. Testing tools may be declared in this section. We have already mentioned testing tools in the previous chapter. In some cases, a customized tool may be required.

Testing requirements have four main sections:

- Testing resource requirements
- Testing tool requirements
- Test cases
- Testing timeline

Scope analysis

Every game has a predefined limited scope. Especially in Android devices, where the variety is maximum, a scope definition is required. Running a game with the same design on all Android platforms is next to impossible.

This section of technical design indicates the probable scope of the game. This may identify the minimum required configuration, recommended configuration, and target configuration to run the game at its maximum performance.

The game scope defines the minimum and maximum range of the hardware platform. Most developers like to minimize the game design scope to target maximum hardware devices. A technical design document is a good reference for developers to get an idea of the performance of the game within the scope.

Risk analysis

A technical design document is made before the production is started, so there are many fields that have to be assumed beforehand. This obviously increases the risk of errors get the solution when the actual problem occurs.

This is the reason risk analysis is mandatory for any technical design standard. The risk may be analyzed in different fields.

change.

So, risk should be calculated to accommodate these changes without affecting the main project pipeline.

of technology should also be addressed. In a common scenario in game development, technology may change during development to increase the game quality.

Change log

It from the
keeping track
of the evolution of the game.

Game design standards

Game design is documented in almost every organization in the gaming industry. This is one of the standards used most often by almost all developers. Technical design is sometimes skipped to save some time, and some designers include the most required segments from a technical document in game design. However, this approach is not recommended.

ng
sections:

- Game overview
- Gameplay details
- Game progression
- Storyboard and game elements
- Level design
- Artificial intelligence
- Art style
- Technical reference
- Change log

Game overview

This section defines the nature of the game along with its target audience. This and feel.

The working title is mentioned beforehand.

o be made.

This section may project a market study to support the game concept and genre chosen for the game.

Gameplay details

ay
is defined in this section. This is one of the most important parts of the game.

The game
rent control

schemes are defined for obvious reasons.

Game progression

Game progression defines the game life cycle and its evolution through time. A game t in time, and this section is responsible for user retention.

Storyboard and game elements

This part of game design defines the background of the game concept. This does not ust have some elements or objects around which the gameplay works.

bstacles, environmental objects, and so on. They are termed as game elements. The reason for running is the background story.

In another example, let's assume a game of Tic-Tac-Toe. A background story is not necessary; however, crosses, circles, and the grid are the elements of the game, which need to be designed and stylized.

Level design

s, introduction, materials or elements, and an objective. More information can be given depending on the game.

Artificial intelligence

Artificial intelligence helps the gameplay to be experienced in a real-time scenario. It ion, collision detection, pathfinding, or anything that determines a state of the game automatically.

Artificial intelligence is mandatory for each and every game. It should imply a domain.

Art style

A game design document also includes the style and direction of the look and feel. The designer may include few references as well. This gives the artist a headstart in thinking about the art direction. Art is the most powerful part of the game to attract users initially.

This section does not include the technical specification for the art. Developers may include a few technical directions here to optimize the asset to be used inside the game.

Technical reference

In this section of a design document, all the technical references are included. For Android game development, this section may include a range of devices with minimum specification, targeting platforms, base graphics engine, development engine, and so on. This is a miniature version of the actual technical design document. When a developer or an organization chooses not to make any separate technical document, they mention all the tech specs in this scope.

Change log

The change log holds a history of changes in the document with versions and dates. This serves the same purpose as any change log documentation.

Other styles and standards

The standard mentioned in the previous sections defines the general process of making a game. We will discuss a few of these processes that are used widely in the game development industry.

racking

system. This may make the development process slower, but effective enough to minimize risk and improve game quality. A few small organizations or individual developers do not follow such processes in order to finalize the product as early as possible.

These styles are opposite to each other, and have different consequences. However, . A quick fix cannot be a permanent solution.

One more commonly used practice is patching code to resolve bugs. This is also extremely vulnerable to threats such as project crash, deadline failure, and creation of a major bug. In game development, the most common problem is a device he cases, it happens because of handling exceptions badly.

It is very necessary to play and understand games to make games. Most game For

Android developers also, it is very good practice to play a lot of games from different small-scale ell. Being an platforms' job of the Android game research and development team.

Different styles for different development engines

We have already discussed a few development tools and engines. The current specific hardware or operating platform. We can find a lot of games that are platform exclusive, but this implies a business decision.

It is quite obvious that the same development is not applicable on every development engine. For example, the development style in native Android will differ from the development style in the Unity3D game engine. The basic reasons are:

- Different programming languages
- Different work principles
- Different target platforms

Different programming languages

Each and every programming language has its own style and structure of me as making games in Android SDK using Java. Developing games using third-party cross-platform engines is also different.

We are not talking about the syntactical difference here. It is about the coding style. Using C++ for Android NDK is different from using C++ for Unreal Engine 4 or des the developer to a different direction of styles to get the best result.

on are being used in the gaming industry. Many of the engines support multiple programming languages to attract maximum developers.

Different work principles

nciples. A developer should be flexible enough to become accustomed to these different systems. There are always different code structures, folder structures, and program hierarchies for different engines.

es come
opers working
on that particular engine will follow the same principle.

engine or Cocos2d-x. Cocos2d-x does not support visual programming, whereas Unreal Engine each must be different despite having the same deployment target.

Different target platforms

style and standard is still different.

hardware is about droid console development is different from Android mobiles.

play session time, control, and look. An average session on a console may last up to 2 hours, whereas mobile session length is almost 5 percent of that. A touch interface is a better interface. Game engine, the style of designing the interface changes for very obvious reasons.

Industry best practices

Although there are plenty of styles and standards out there, most developers like to maintain some common standards to create stability in the game development procedure. Let's discuss some of these areas of standards commonly practiced by the industry:

- Design standards
- Programming standards
- Testing standards

Design standards

The goal is to make it properly documented along with scope for improvement. The document concept is, standard.

e game; this causes serious delay in production time. However, it should always have a limited scope to improve the production time over time with ideas.

son.

This should also include probable target hardware platforms.

esent day.

So, most Android developers mainly have their focus on mobile games. However, designers should always leave scope for the game to be deployed on other platforms, such as wearables, consoles, and so on.

Programming standards

Programming is the execution of the design. It is the most significant part of the production of any game. A standard piece of code should be readable, modular, es:

ustry,

ing

standards have changed a lot. Previously, it was common practice to use `m_` and `l_` as prefire. There

were a few other notations such as `i`, `f`, and `b` and so on to indicate variable types.

Modern day standards follow mainly the Camel and Pascal casing system for their naming convention. Common practice is to use Pascal casing for all classes, interfaces, enums, events, structures, namespaces, and method names, and other elements should use the Camel casing system.

Camel casing in programming language means that the first letter of a name should be in lowercase, which is specifically **Lower CamelCase**. The Pascal casing system states that the first letter should be capitalized, which is termed **Upper CamelCase**.

d or the

mon

industry practice says that the number of arguments should be within eight, and the number of letters per line of coding should not exceed 20.

The reason for this manual limitation is to have less complexity and better readability of the code. For the same reason, a method body should be limited to 200 lines, and a split class structure is always preferred.

Testing standards

Testing also checks the development standard, and it also ensures the quality of the game.

automated
ing code
for checking the core development. This part is called the test code, which must not
rs to carry
the user point
of view.

Most game development companies follow a checklist for the testing procedure. This checklist often contains defined test cases. Test cases are mainly defined by the developer and designer, and testers need to execute these cases. We will discuss testing in detail in a later chapter.

Summary

Any software development must follow a certain protocol and standard. Game sustain
for a longer period of time. The modern age Android game life cycle includes
. For an
or a long
period time, which is a very common scenario in the game industry.

elopers
and be flexible enough to accommodate new changes for updates to the game.

droid
game development industry. Game developers should follow the game development
e it easy
ate a
formation

to programmers and game engineers. A specific development process in an organization defines and maintains development standards. Programmers must write code in modules to avoid future changes and to increase the reusability of codes. A proper naming convention always helps in understanding the code better, and prepares it for easy editing and reuse.

play and
enjoy a lot of games.

5

Understanding the Game Loop and Frame Rate

he consequence. A game cannot be made without a defined game loop, and the performance cannot be judged without measuring the frame rate.

These two aspects of game development are common throughout any game p vary across different devices, and there might be different scales to measure frame rates across different platforms.

opers only. However, in most game engines, the loop is already defined with all the necessary controls and scope.

evelopment through the following topics:

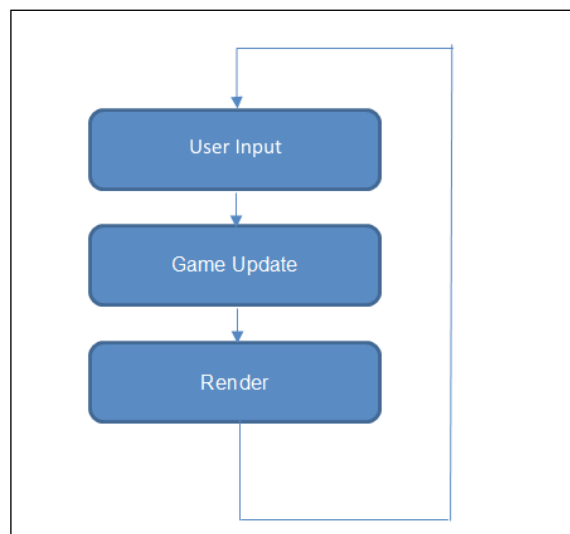
- Introduction to the game loop
- Creating a sample game loop using the Android SDK
- Game life cycle
- Game update and user interface
- Interrupt handling
- General idea of a game state machine
- The FPS system
- Hardware dependency
- Balance between performance and memory
- Controlling FPS

Introduction to the game loop

ndering are
me loop is the
most important part of running a game with frame rate control.

A typical game loop has three steps:

1. User input
2. Game update
3. Rendering



A simple game loop

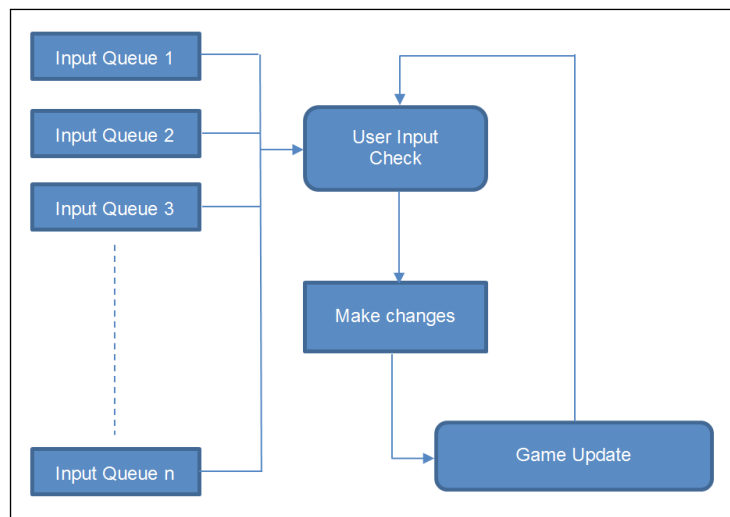
User input

This section checks the UI system of the game for any external input that has been given to the game. It sets the required changes to be made in the game for the next varies the
fferent input
types to make a standard.

er-given
ors the
input system, whether an event has occurred or not.

A user can trigger any event at any point of time during gameplay when an active game loop is running. Normally, there are queues maintained by the input system. touch, key press, sensor reading, and so on.

owing the loop sequence. If it finds any event in the queue, it makes the required changes that will have an impact on the next update call in the game loop:



User input working principle

Game update

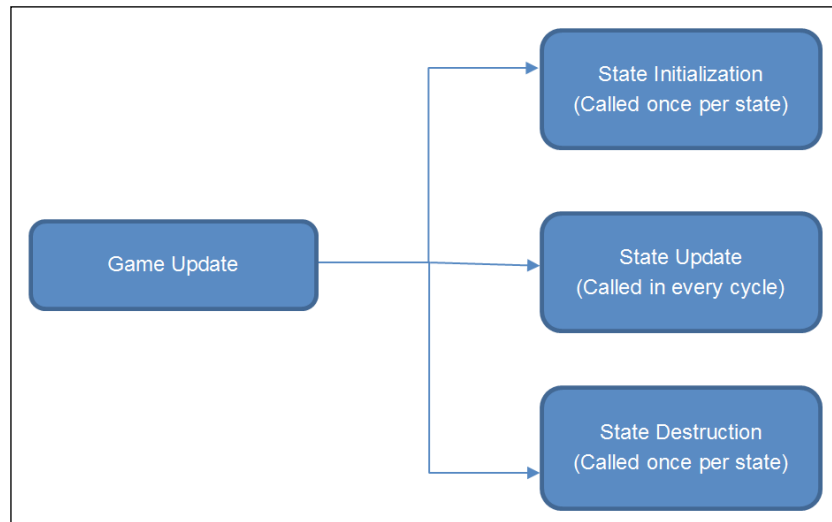
tion of the game loop. This section is also responsible for running the game logic, changes in e.

the main used game program structure in the previous section.

Any game runs a particular state at a time. The state can be updated by either user update cycle frame by frame.

State update

is also
on happens
once per state, and state update can be called once per game cycle.



State update call flow

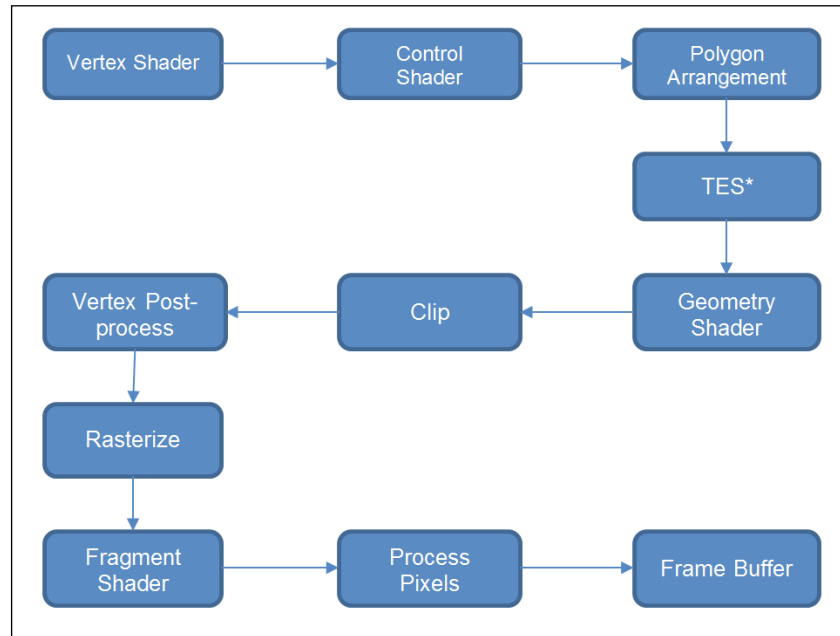
Rendering frames

The rendering section inside a game loop is responsible for setting the rendering p.

line. The
developer could manipulate and set each and every vertex. The modern age game
aphics
at a very high
ices.

control,
ns take
the most time to execute from the processing point of view.

Typical Android graphics rendering follows the OpenGL pipeline:



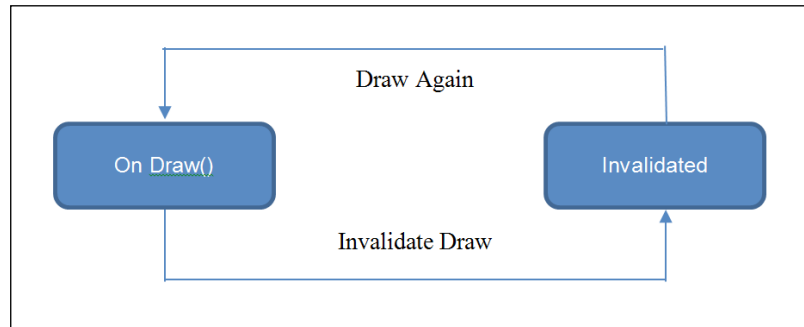
Creating a sample game loop using the Android SDK

Android SDK development starts with an activity, and the game runs on single or multiple views. Most of the time, it is considered to have a single view to run gameplay.

Unfortunately, the Android SDK does not provide a predefined game loop. It remains the same.

In the Android SDK library, the `View` class contains an abstract method `onDraw()` in which every possible rendering call is queued. This method is called upon any

The logic is as follows:



Let's have a look at a basic game loop created with Android view. Here, a custom view is extended from the Android View:

```
/*Sample Loop created within OnDraw() on Canvas
 * This loop works with 2D android game development
 */
@Override
public void onDraw(Canvas canvas)
{
    //If the game loop is active then only update and render
    if(gameRunning)
    {
        //update game state
        MainGameUpdate();

        //set rendering pipeline for updated game state
        RenderFrame(canvas);
        //Invalidate previous frame, so that updated pipeline can be
        // rendered
        //Calling invalidate() causes recall of onDraw()
        invalidate();
    }
    else
    {
        //If there is no active game loop
        //Exit the game
        System.exit(0);
    }
}
```

In the current age of Android game development, developers use `SurfaceView` instead of `View`. `SurfaceView` is inherited from `View` and more optimized for games made with `Canvas`. In this case, a customized view is extended from `SurfaceView` and implements the `SurfaceHolder.Callback` interface. In this scenario, three methods are overridden:

```
/* Called When a surface is changed */
@Override
public void surfaceChanged(SurfaceHolder holder, int format, int
    width, int height)
{
}
/* Called on create of a SurfaceView */
@Override
public void surfaceCreated(SurfaceHolder holder)
{
}
/* Called on destroy of a SurfaceView is destroyed */
@Override
public void surfaceDestroyed(SurfaceHolder holder)
{
}
```

time.

That's the reason the `surfaceChanged` method should have an empty body to function as a basic game loop.

We need to create a customized game thread and override the `run()` method:

```
public class BaseGameThread extends Thread
{
    private boolean isGameRunning;
    private SurfaceHolder currentHolder;
    private MyGameState currentState;
    public void activateGameThread(SurfaceHolder holder, MyGameState
        state)
    {
        currentState = state;
        isGameRunning = true;
        currentHolder = holder;
        this.start();
    }
}
```



```
@Override
public void run()
{
    Canvas gameCanvas = null;
    while(isGameRunning)
    {
        //clear canvas
        gameCanvas = null;
        try
        {
            //locking the canvas for screen pixel editing
            gameCanvas = currentHolder.lockCanvas();
            //Update game state
            currentState.update();
            //render game state
            currentState.render(gameCanvas);
        }
        catch(Exception e)
        {
            //Update game state without rendering (Optional)
            currentState.update();
        }
    }
}
```

SurfaceView class:

```
public myGameCanvas extends SurfaceView implements SurfaceHolder
{
    //Declare thread
    private BaseGameThread gameThread;
    private MyGameState gameState;
    @Override
    public void surfaceCreated(SurfaceHolder holder)
    {
        //Initialize game state
        gameState = new MyGameState();
        //Instantiate game thread
    }
}
```

```

        gameThread = new BaseGameThread();
        //Start game thread
        gameThread. activateGameThread(this.getHolder(),gameState);
    }

    @Override
    public void surfaceChanged(SurfaceHolder holder, int format, int
        width, int height)
    {
    }

    @Override
    public void surfaceDestroyed(SurfaceHolder holder)
    {
    }
}

```

asic
prefer
ther
s chapter.

Another part of this game loop is **frames per second (FPS)** management. One of the most common mechanisms is to use `Thread.sleep()` for such a calculated time that the loop executes at a fixed rate. Some developers create two types of update mechanism: one based on FPS and another based on per frame without delay.

Mostly, physics-based games need an update mechanism that follows a real-time interval to function uniformly across all devices.

For small-scale development, few developers in the industry follow the first current
t dependent
on fixed FPS.

Game life cycle

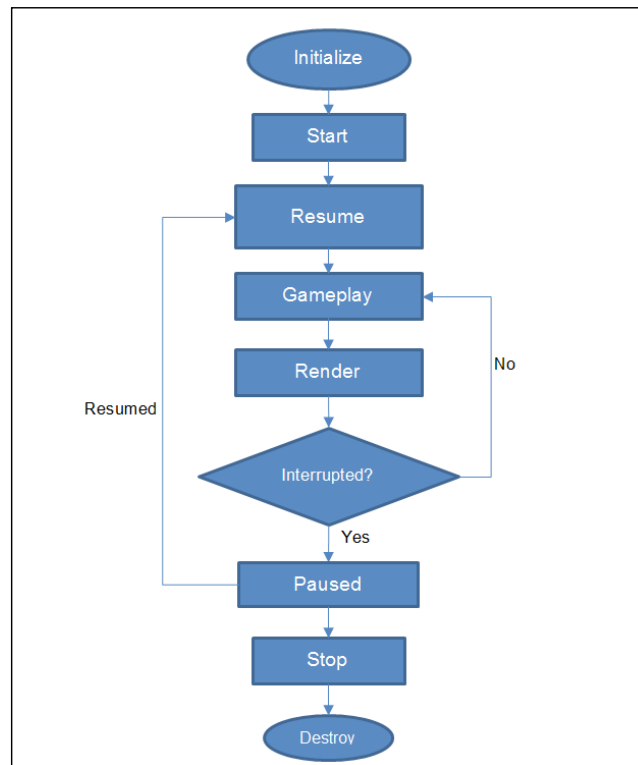
life cycle,
s with
have
algorithms or artificial intelligence that is capable of interfering with the main game cycle.

An Android game is initialized with an activity. The `onCreate()` method is used for the game loop can then be interrupted by an external interrupt.

the current state, it should be easy to return to the last state.

Game update and user interface

We have already covered a few update and interface mechanisms previously. A running game state can be changed by user input or internal AI algorithms:



Mostly, game update is called once per frame or once after a fixed time interval. Either way, an algorithm does its job to change the game state. You have learned about user input queues. On each game loop cycle, the input queues are being checked.

```
/* import proper view and implement touch listener */
public class MyGameView extends View implements
    View.OnTouchListener
/* declare game state */
private MyGameState gameState;
/* set listener */
public MyGameView (Context context)
{
    super(context);
    setOnTouchListener(this);
    setFocusableInTouchMode(true);
    gameState = new MyGameState();
}

/* override onTouch() and call state update on individual touch
events */
@Override
public boolean onTouch(View v, MotionEvent event)
{
    if(event.getAction() == MotionEvent.ACTION_UP)
    {
        //call changes in current state on touch release
        gameState.handleInputTouchRelease((int)event.getX(),
            (int)event.getY());
        return false;
    }
    else if(event.getAction() == MotionEvent.ACTION_DOWN)
    {
        //call changes in current state on touch begin
        gameState.handleInputTouchEngage((int)event.getX(),
            (int)event.getY());
    }
    else if(event.getAction() == MotionEvent.ACTION_MOVE)
    {
        //call changes in current state on touch drag
        gameState.handleInputTouchDrag((int)event.getX(),
            (int)event.getY());
    }
    return true;
}
```

```
Point touchBegin = null;
Point touchDragged = null;
Point touchEnd = null;

@Override
public boolean onTouch(View v, MotionEvent event)
{
    if(event.getAction() == MotionEvent.ACTION_UP)
    {
        touchEnd = new Point((int)event.getX(), (int)event.getY());
        return false;
    }
    else if(event.getAction() == MotionEvent.ACTION_DOWN)
    {
        touchBegin = new Point((int)event.getX(), (int)event.getY());

    }
    else if(event.getAction() == MotionEvent.ACTION_MOVE)
    {
        touchDragged = new Point((int)event.getX(), (int)event.getY());

    }
    return true;
}

/* declare checking input mechanism */
private void checkUserInput()
{
    if(touchBegin != null)
    {
        //call changes in current state on touch begin
        gameState.handleInputTouchEngage (touchBegin);
        touchBegin = null;
    }

    if(touchDragged != null)
    {
        //call changes in current state on touch drag
    }
}
```

```
        gameState.handleInputTouchDrag (touchDragged);
        touchDragged = null;
    }

    if(touchEnd != null)
    {
        //call changes in current state on touch release
        gameState.handleInputTouchRelease (touchEnd);
        touchEnd = null;
    }
}

/* finally we need to invoke checking inside game loop */
@Override
public void onDraw(Canvas canvas)
{
    //If the game loop is active then only update and render
    if(gameRunning)
    {
        //check user input
        checkUserInput();
        //update game state
        MainGameUpdate();

        //set rendering pipeline for updated game state
        RenderFrame(canvas);
        //Invalidate previous frame, so that updated pipeline can be
        // rendered
        //Calling invalidate() causes recall of onDraw()
        invalidate();
    }
    else
    {
        //If there is no active game loop
        //Exit the game
        System.exit(0);
    }
}
```

The same process can be repeated for the SurfaceView game loop approach as well.

Interrupt handling

is necessary

to pause every running thread and save the current state of the game to ensure that it resumes properly.

In Android, any interrupt triggers from `onPause()`:

```
@Override
protected void onPause()
{
    super.onPause();
    // pause and save game loop here
}
// When control is given back to application, then onResume() is //
called.
@Override
protected void onResume()
{
    super.onResume();
    //resume the game loop here
}
```

Now, we need to change the class where the actual game loop is running.

Then, put a
this variable:

```
private static boolean gamePaused = false;
@Override
public void onDraw(Canvas canvas)
{
    if(gameRunning && ! gamePaused)
    {
        MainGameUpdate();
        RenderFrame(canvas);

        invalidate();
    }
    else if(! gamePaused)
    {
        //If there is no active game loop
        //Exit the game
    }
}
```

```
        System.exit(0);
    }
}

public static void enableGameLoop(boolean enable)
{
    gamePaused = enable;
    if(!gamePaused)
    {
        //invalidation of previous draw has to be called from static
        // instance of current View class
        this.invalidate();
    }
    else
    {
        //save state
    }
}
```

General idea of a game state machine

me state

techniques,

this was a typical linear control flow. However, in modern development processes, it can be parallel control running in multiple threads. In the old architecture of game used to nagement.

However, even in modern development, many developers still prefer to use a single tools

and advanced scripting language, most game developers now use a virtual parallel processing system.

One of the processes of a simple game state machine is to create a common state asy to manage the state inside the game loop.

Let's see a loop of a simple game state machine manager. This manager should conduct four main functionalities:

- Creating the state
- Updating the state
- Rendering the state
- Changing the state

An example implementation might look like this:

```
public class MainStateManager
{
    private int currentStateId;
    //setting up state IDs
    public Interface GameStates
    {
        public static final int STATE_1 = 0;
        public static final int STATE_2 = 1;
        public static final int STATE_3 = 2;
        public static final int STATE_4 = 3;
    }

    private void initializeState(int stateId)
    {
        currentStateId = stateId;
        switch(currentStateId)
        {
            case STATE_1:
                // initialize/load state 1
                break;
            case STATE_2:
                // initialize/load state 2
                break;
            case STATE_3:
                // initialize/load state 3
                break;
            case STATE_4:
                // initialize/load state 4
                break;
        }
    }
}

/*
 * update is called in every cycle of game loop.
 * make sure that the state is already initialized before updating
   the state
 */
private void updateState()
{
    switch(currentStateId)
    {
```

```
        case STATE_1:
            // Update state 1
            break;
        case STATE_2:
            // Update state 2
            break;
        case STATE_3:
            // Update state 3
            break;
        case STATE_4:
            // Update state 4
            break;
    }
}
/*
 * render is called in every cycle of game loop.
 * make sure that the state is already initialized before updating
   the state
 */
private void renderState()
{
    switch(currentStateId)
    {
        case STATE_1:
            // Render state 1
            break;
        case STATE_2:
            // Render state 2
            break;
        case STATE_3:
            // Render state 3
            break;
        case STATE_4:
            // Render state 4
            break;
    }
}
/*
 * Change state can be triggered from outside of manager or from
   any other state
 * This should be responsible for destroying previous state and
   free memory and initialize new state
 */
```

```
public void changeState(int nextState)
{
    switch(currentStateId)
    {
        case STATE_1:
            // Destroy state 1
            break;
        case STATE_2:
            // Destroy state 2
            break;
        case STATE_3:
            // Destroy state 3
            break;
        case STATE_4:
            // Destroy state 4
            break;
    }
    initializeState(nextState);
}
```

In some cases, developers pass the input signal to a particular state through the state manager as well.

The FPS system

The game

he higher

the FPS of the game, the better. The FPS of a game is dependent on the processing time for instructions and rendering.

sample

implementation of FPS management inside a game loop:

```
long startTime;
long endTime;
public final int TARGET_FPS = 60;

@Override
public void onDraw(Canvas canvas)
{
    if(isRunning)
    {
```

```

        startTime = System.currentTimeMillis();
        //update and paint in game cycle
        MainGameUpdate();

        //set rendering pipeline for updated game state
        RenderFrame(canvas);

        endTime = System.currentTimeMillis();
        long delta = endTime - startTime;
        long interval = (1000 - delta)/TARGET_FPS;

        try
        {
            Thread.sleep(interval);
        }
        catch(Exception ex)
        {}
        invalidate();
    }
}

```

In the preceding example, we first noted the time before execution (`startTime`) of the loop and then noted down the time after the execution (`endTime`). We then calculated the time taken for execution (`delta`). We already know the amount of time (`interval`) time, we put the game thread to sleep before it executes again. This can be applied to a different game loop system as well.

While using `SurfaceView`, we can declare the FPS system inside the game loop in the `run()` method:

```

long startTime;
long endTime;
public final int TARGET_FPS = 60;
@Override
public void run()
{
    Canvas gameCanvas = null;
    while(isGameRunning)
    {
        startTime = System.currentTimeMillis();
        //clear canvas
    }
}

```

```
gameCanvas = null;
try
{
    //locking the canvas for screen pixel editing
    gameCanvas = currentHolder.lockCanvas();
    //Update game state
    currentState.update();
    //render game state
    currentState.render(gameCanvas);
    endTime = System.currentTimeMillis();
    long delta = endTime - startTime;
    long interval = (1000 - delta)/TARGET_FPS;

    try
    {
        Thread.sleep(interval);
    }
    catch(Exception ex)
    {}
}
Catch(Exception e)
{
    //Update game state without rendering (Optional)
    currentState.update();
}
}
```

In this process, we capped the FPS count and tried to execute the game loop on the predefined FPS. A major drawback in this system is this mechanism massively depends on hardware configuration. For a slow hardware system, which is incapable of running the loop on the predefined FPS, this system has no effect. This is because of the same cycle.

Hardware dependency

We have discussed earlier that hardware configuration plays a major role in the FPS of the game on the target FPS.

Let's list the tasks that take most of the processing time for a game:

- Display or rendering
- Memory load/unload operations
- Logical operations

Display or rendering

Display processing depends mostly on the graphics processor and what all needs to become
g takes
time.

There were times when running a game with a frame rate of 12 was difficult.
run
on a frame rate of 60. It is only a matter of hardware quality.

A large display requires a good amount of cache memory. So, for example,
hardware with a large and dense display and with low cache memory is incapable
of maintaining a good display quality.

Memory load/unload operations

Memory is a hardware component of a system. Again, it takes more time to interact
when we
allocate memory, deallocate memory, and read or write an operation.

important:

- Heap memory
- Stack memory
- Register memory
- ROM

Heap memory

Heap memory is user-defined manually managed memory. This memory has to be
allocated manually and freed manually as well. In the case of Android, the garbage
collector is responsible for freeing memory, which is flagged as non-referenced. This
memory location is the slowest in the random access memory category.

Stack memory

This segment of memory is used for elements that are declared inside a method. Allocation and deallocation of this memory segment is automatically done by the program interpreter. This memory segment works only for local members.

Register memory

data for the
higher
d faster.

ROM

Read-only memory (ROM) is permanent memory. Especially in game development, the load/unload operation of those assets. A program needs to load the necessary data onto the RAM from the ROM. So, having faster ROM helps achieve better FPS during the load/unload operation.

Logical operations

Developers should define the instructions in such a way that they can use hardware in the most efficient way. In technical terms, each and every instruction goes in one clock cycle.

n:

```
char[] name = "my name is android";
for(int i = 0; i < name.length; i ++){
    //some operation
}
```

Calling `length` and using a post increment operator every time increases the time.

Now, look at this code:

```
char[] name = "my name is android";
int length = name.length;
for(int i = 0; i < length; ++ i){
    //some operation
}
```

This code executed the same task; however, the processing overhead is reduced a lot in this approach. The only compromise this code made is blocking memory for one integer variable and saving a lot of nested tasks related to `length`.

directly
eloper,
as is shown in the previous example.

Every processor has a mathematical processing unit. The power of the processor varies from one processor to another. So, developers always need to check the mathematical expression to know whether it can be simplified or not.

Balance between performance and memory

lopers
lance
between performance and memory.

Loading or unloading any asset from ROM to RAM takes time, so it is recommended
eration
affects FPS significantly.

the
r should
group assets. Small assets can be loaded in the game running the state only in
required cases.

Sometimes, many developers preload all the assets and use it from cache. This
in a
game if an
interrupt occurs. The Android OS is fully authorized to clear memory occupied by
oes to
y is not
for a new
application.

ccording to
game states.

Controlling FPS

We have already seen some ways of defining the FPS system. We have already discussed the major drawback of the system as well. So, we can manipulate the oop cycle:

```
long startTime;
long endTime;
public static int ACTUAL_FPS = 0;

@Override
public void onDraw(Canvas canvas)
{
    if(isRunning)
    {
        startTime = System.currentTimeMillis();
        //update and paint in game cycle
        MainGameUpdate();

        //set rendering pipeline for updated game state
        RenderFrame(canvas);

        endTime = System.currentTimeMillis();
        long delta = endTime - startTime;
        ACTUAL_FPS = 1000 / delta;
        invalidate();
    }
}
```

FPS to

60. Otherwise, the game can be manipulated through actual FPS:

```
long startTime;
long endTime;
public final int TARGET_FPS = 60;
public static int ACTUAL_FPS = 0;

@Override
public void onDraw(Canvas canvas)
{
    if(isRunning)
    {
        startTime = System.currentTimeMillis();
        //update and paint in game cycle
```

```
MainGameUpdate();

//set rendering pipeline for updated game state
RenderFrame(canvas);

endTime = System.currentTimeMillis();
long delta = endTime - startTime;

//hybrid system begins
if(delta < 1000)
{
    long interval = (1000 - delta)/TARGET_FPS;
    ACTUAL_FPS = TARGET_FPS;
    try
    {
        Thread.sleep(interval);
    }
    catch(Exception ex)
    {}
}
else
{
    ACTUAL_FPS = 1000 / delta;
}
invalidate();
}
```

Summary

The game loop is mainly a logical approach for game development. In many cases, developers do not opt for such a mechanism. Some games may be typically interactive and have no algorithm that runs continuously. In such cases, the game loop may not be needed. Game states can be updated as per input given to the gaming system.

ial
ective
of game design.

rs

any game

ates.

The idea and concept of game loop and state management may change as per the game requirement.

re using

should not affect the game performance and FPS. Besides that, developers need to maintain the readability and flexibility of code. Some approaches may consume more memory and run faster and vice versa. Android has various sets of hardware confin

all hardware. Finally, balancing between memory and performance is the key to creating better games.

We will have a deep look at performance and memory management in later different

perspectives, such as 2D/3D games, VR games, optimization techniques, and more.

6

Improving Performance of 2D/3D Games

-white

pixel

graphics. Times have changed now. 3D games are running on handhelds with ease.

dc core 3D

game, 2D assets are mandatory. Few games are fully 2D.

We will discuss the performance of 2D and 3D games here with the help of the following topics:

- 2D game development constraints
- 3D game development constraints
- The rendering pipeline in Android
- Rendering through OpenGL
- Optimizing 2D assets
- Optimizing 3D assets
- Common game development mistakes
- 2D/3D performance comparison

2D game development constraints

follows:

- 2D art assets
- 2D rendering system
- 2D mapping
- 2D physics

2D art assets

Art asset constraints are mainly limited to graphical or visual assets, which include images, sprites, and fonts. It is not difficult to understand that a larger asset will take more time to process and render than a smaller asset, resulting in less performance quality.

Sets of 2D art assets

of assets in

Android game development. This is the reason most Android game developers choose high-resolution assets as their base build. This normally performs well for high-configuration hardware platforms, but does not provide quality performance on low-configuration devices. Many developers opt for the option of porting for the project.

Same asset set for multiple resolutions

Many times, developers choose to ignore a set of hardware platforms. Mostly, in resolution art assets and fit them into lower resolution devices by scaling down. Nowadays, most hardware platforms have better RAM. Hence, this process has become convenient for developers.

Number of assets drawn on screen

Game performance does not always depend on the asset size; it also depends on the sheet has evolved to reduce the number of drawing elements on screen.

art asset. As

the number of assets increases, it takes more such draw instructions to complete in the processor, and the game performance becomes poor.

it takes only

one draw instruction to render all the assets of the sprites. However, the physical size of the sprite sheet is restricted. The maximum size varies for different devices with the safest in the current scenario.

Use of font files

Almost every game uses custom or special fonts other than the default system font of Android. In those cases, the font source file has to be included in the game build. them here:

- Sprite font
- Bitmap font
- TrueType font

Sprite font

This is a typical old school technique but is still effective in some cases. The characters are mapped within a data file. This mapping is used to clip each character and form words accordingly.

Here are some advantages of this font:

- Developers have total control of mapping
- Character stylization can be customized as per requirement
- Fast processing speed can be achieved; however, it will depend on development efficiency

Here are some disadvantages of this font:

- They increase development overhead
- The system efficiency entirely depends on the developer's skill set
- It is very difficult to map characters in the case of multi-language support
- Any change takes a lot of iteration to achieve perfection

This style is not usually used nowadays as we have many designer and stylish fonts available.

Bitmap font

with a predefined one or more sprite sheets with one data file. The working principle of bitmap font is like TrueType fonts with a bit of stylization directly from the TrueType font.

Here are some advantages of this font:

- It is compatible with any existing codebase, irrespective of the rendering framework, whether it is OpenGL, DirectX, Direct Draw, or GDI+
- It is easy to integrate
- It can manipulate the style of the existing TrueType font

Here are some disadvantages of this font:

- The same disadvantages of the sprite font are applicable here, only with less development overhead
- Scaling up the bitmap font results in blurry output

TrueType font

, including Android. It is the fastest way to integrate various fonts in games.

Here are some advantages of this font:

- Universal font style
- Maximum platform support
- Easy multi-language implementation
- This is a vector font, so it has no scaling issue
- Easy special character availability

Here are some disadvantages of this font:

- Using this font style may cost a few kilobytes extra to the game
- Not all scripting languages are supported by TTF

2D rendering system

Android provides a scope to render 2D assets onto the canvas through an API framework. Canvas can be used with Drawable objects in View or SurfaceView.

the graphical objects can be drawn. Draw on the canvas happens within the `onDraw()` callback method. The developer just needs to specify graphical objects along with their position on the canvas.

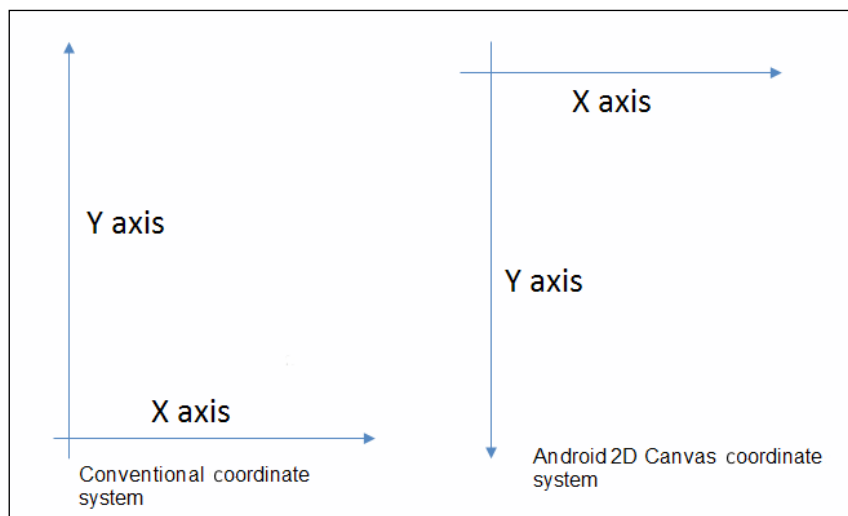
Canvas itself has a set of default drawing methods to render almost each type of graphical objects. Here are some examples:

- The `drawBitmap()` method is used to draw image objects in the bitmap format. However, images need not be in bitmap format.
- The `drawRect()` and `drawLine()` methods are used to draw primitive shapes on the canvas.
- The `drawText()` method can be used to render text on canvas using a specific font style.

Canvas can be used within a view in the Android architecture.

2D mapping

2D mapping is based on a simple 2D coordinate system. The only difference is the opposite *y* axis in comparison with the conventional coordinate system:



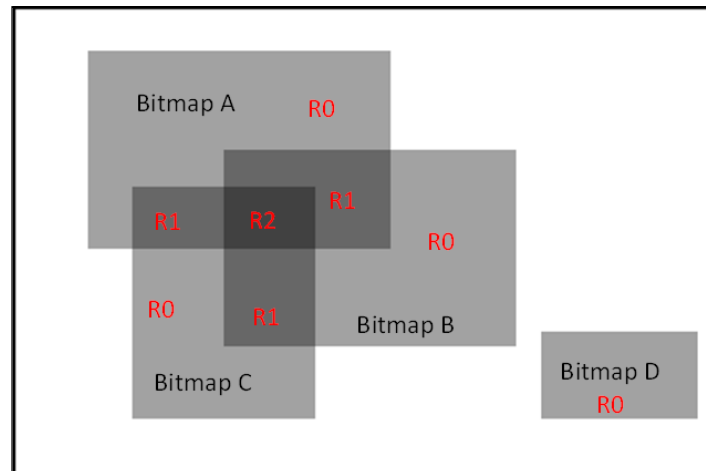
s. All the
irect effect
pers are
nd they
reverse the vertical axis to render it on the canvas. This requires some additional
calculation.

There is one more performance constraint regarding the 2D rendering systems. A common development approach across the world is to have a minimum set of graphic assets and use them as much as possible. Often, this leads to rendering the same pixel multiple times. This affects the processing speed and hence the FPS.

For example, bitmap **A**, bitmap **B**, bitmap **C**, and bitmap **D** are being rendered on a canvas in such a way that **A**, **B**, and **C** overlap each other, and **D** remains separate. The following happens:

- Pixels in the region **R0** where only one bitmap is drawn will be rendered once
- Pixels in region **R1** where two bitmaps are overlapping will be rendered twice
- Pixels in region **R2** where three bitmaps are overlapping will be rendered three times

This is shown here:



Now, in regions **R1** and **R2**, all the pixels are rendered multiple times. In this system, the pixel data information will append to the previous data, resulting in the final pixel value. In this system, the processing overhead increases. Hence, performance decreases.

Even today, it is a common practice for 2D game programming. The reasons are as follows:

- Transparency blending
- Modular graphical assets

- Low build size
- Easy construction of screens by overlapping multiple assets

Sometimes, there may be a scenario where a device has a very low-performing graphics processor, and rendering the same pixel multiple times has a major impact t.

set in which

object is drawn

on the screen only once. It prevents the following issues:

- Flickering of screen
- Multiple draws of one pixel
- Tearing of assets

2D physics

2D physics takes only the x - y plane into consideration for all the calculations. There are plenty of 2D physics engines available in market. **Box2D** is the most popular one. A physics engine consists of every mechanism and calculation of real-time physics.

Real-time physics calculation is much complicated than is required in games. Let's discuss a few available physics engines.

Box2D

Box2D is an open source physics engine based on C++. It consists of almost every aspect of solid physics that can be used in various games. A few of its mentionable features are as follows:

- Dynamic collision detection of rigid bodies
- Collision state callbacks, such as collision enter, exit, stay, and so on
- Polygonal collision
- Vertical, horizontal, and projectile motion
- Friction physics
- Torque and momentum physics
- Gravity effects based on pivot point and joints

LiquidFun

ngine is

to cover

the liquid physics formula and mechanism. LiquidFun can be used for Android, iOS, ture of

Box2D, along with liquid particle physics. This includes the following:

- Wave simulation
- Liquid fall and particle simulation
- Liquid stir simulation
- Solid and liquid dynamic collision
- Liquid mixing

Performance impact on games

Collision detection is a costly process. Multi-edge and polygonal collisions increase the process overhead. The number of rigid bodies and collision surfaces have the maximum impact on performance. This is why liquid physics is slower than solid physics.

Let's have a look at the major impacts:

- Each transformation of any rigid body requires a refresh on the collision check of the entire system
- The physics engine is responsible for repetitive transform change, which is responsible for heavy processes

sics engine.

lways

zation is

required for games.

2D collision detection

Most games use the box-colliding system to detect most collisions. Rectangular collision detection is the cheapest possible method, which can be used inside games to detect collisions.

Sometimes, triangular and circular collision detection is also used for 2D games for collision detection accuracy. There needs to be a balance of using such methods.

can opt for
any of these systems:

- Considering each circle a rectangle and detecting the collision between them
- Considering one circle a rectangle and detecting the collision between the circle and rectangle
- Applying the actual circular collision detection method

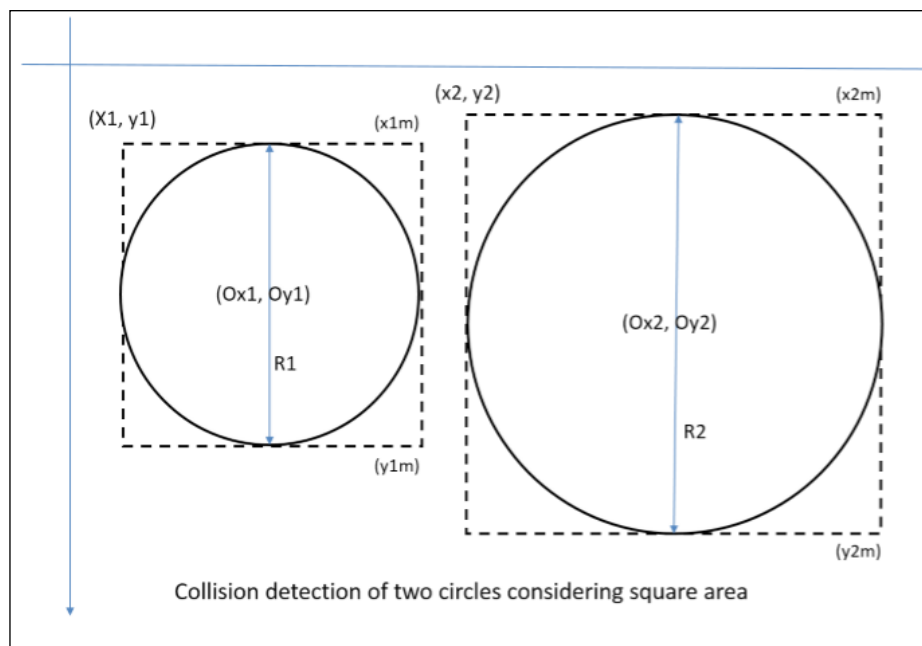
Let's consider two circles having origins $O1$ and $O2$ and diameters $R1$ and $R2$:

$O1$ is located at $(Ox1, Oy1)$

$O2$ is located at $(Ox2, Oy2)$

Rectangle collision

like this:



Rectangular collision detection refers to this formula.

Input feed will be as follows:

$xMin1 = x1$ (minimum co-ordinate on x axis of first rectangle)

$y_{Min1} = y1$ (minimum co-ordinate on y axis of the first rectangle)

$x_{Max1} = x1m$ (maximum co-ordinate on x axis of the first rectangle)

$y_{Max1} = y1m$ (maximum co-ordinate on y axis of the first rectangle)

$x_{Min2} = x2$ (minimum co-ordinate on x axis of the second rectangle)

$y_{Min2} = y2$ (minimum co-ordinate on y axis of the second rectangle)

$x_{Max2} = x2m$ (maximum co-ordinate on x axis of the second rectangle)

$y_{Max2} = y2m$ (maximum co-ordinate on y axis of the second rectangle)

In the given circumstances, we will have the following:

$$x1 = Ox1 - (R1 / 2)$$

$$y1 = Oy1 - (R1 / 2)$$

$$x1m = Ox1 + (R1 / 2) = x1 + R1$$

$$y1m = Oy1 + (R1 / 2) = y1 + R1$$

$$x2 = Ox2 - (R2 / 2)$$

$$y2 = Oy2 - (R2 / 2)$$

$$x2m = Ox2 + (R2 / 2) = x2 + R2$$

$$y2m = Oy2 + (R2 / 2) = y2 + R2$$

The condition for colliding or not colliding these two rectangles will be as follows:

```
if( x1m < x2 )
{
    // Not Collide
}
else if( y1m < y2 )
{
    // Not collide
}
else if( x1 > x2m )
{
    //Not collide
```

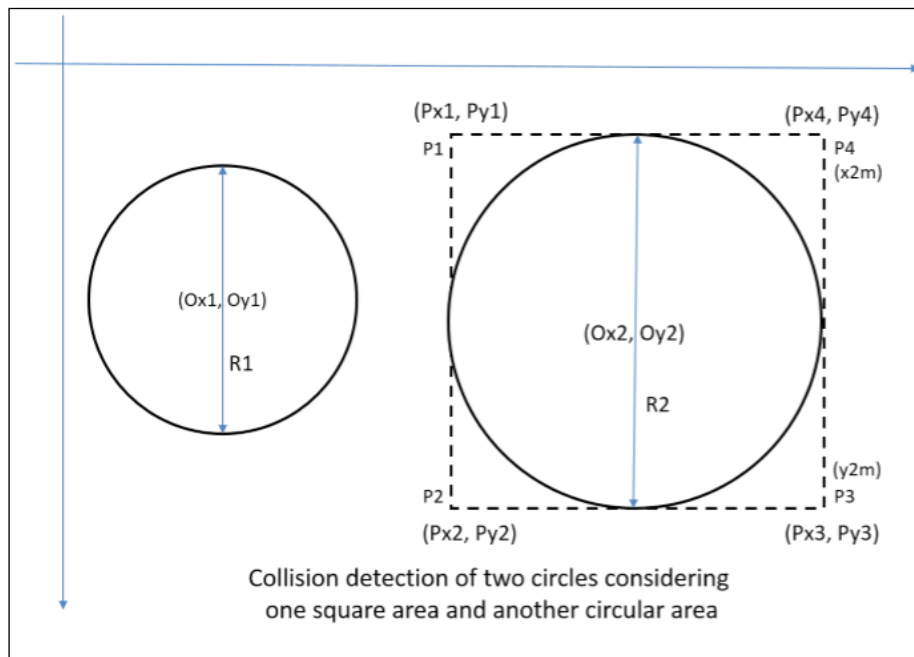
```

}
else if( y1 > y2m )
{
    //Not collide
}
else
{
    //Successfully collide
}

```

Rectangle and circle collision

Now, considering only the second circle as a rectangle, we will have this:



As we have already discussed the general idea of the coordinate system for the same system, we can directly derive the values:

$$Px1 = Ox2 - (R2 / 2)$$

$$Py1 = Oy2 - (R2 / 2)$$

$$Px2 = Ox2 + (R2 / 2)$$

$$Py2 = Oy2 + (R2 / 2)$$

$$Px3 = Ox2 + (R2 / 2)$$

$$Py3 = Oy2 + (R2 / 2)$$

$$Px4 = Ox2 + (R2 / 2)$$

$$Py4 = Oy2 - (R2 / 2)$$

$$x2m = Ox2 + (R2 / 2) = x2 + R2$$

$$y2m = Oy2 + (R2 / 2) = y2 + R2$$

$$radius1 = (R1 / 2)$$

$$1 - Oy1) * (Py1 - Oy1)))$$

$$2 - Oy1) * (Py2 - Oy1)))$$

$$3 - Oy1) * (Py3 - Oy1)))$$

$$4 - Oy1) * (Py4 - Oy1)))$$

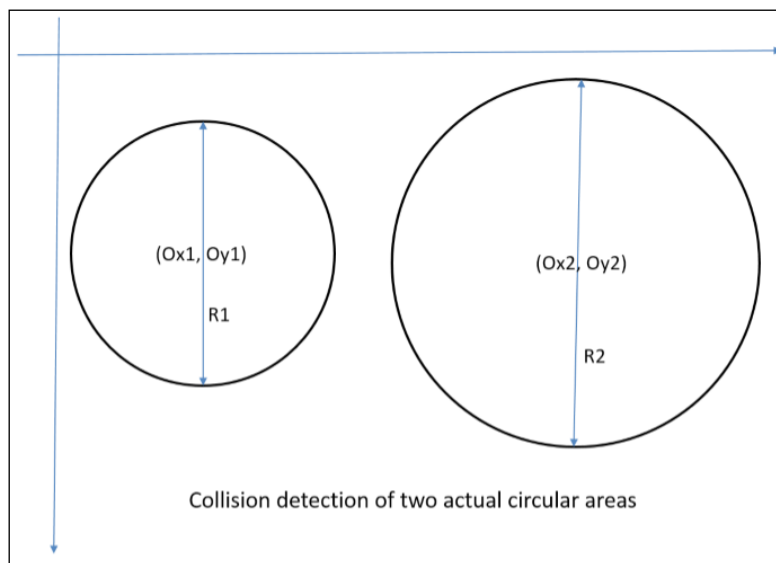
The colliding and non-colliding condition would be as follows:

```
if ( (Ox1 + radius1) < x2 )
{
    //Not collide
}
else if ( Ox1 > x2m )
{
    //Not collide
}
else if ( (Oy1 + radius1) < y2 )
{
    //Not collide
}
else if ( Oy1 > y2m )
{
    //Not collide
}
else
{
    if (distanceP1 <= radius1)
    {
        //Successfully collide
    }
    else if (distanceP2 <= radius1)
    {
        //Successfully collide
    }
}
```

```
else if (distanceP3 <= radius1)
{
    //Successfully collide
}
else if (distanceP4 <= radius1)
{
    //Successfully collide
}
else if ( Ox1 >= Px1 && Ox1 <= x2m &&
(Oy1 + radius1) >= Py1 && (Oy1 <= y2m))
{
    //Successfully collide
}
else if ( Oy1 >= Py1 && Oy1 <= y2m &&
(Ox1 + radius1) >= Px1 && (Ox1 <= x2m))
{
    //Successfully collide
}
else
{
    //Not collide
}
```

Circle and circle collision

circle collision:



Logically, this is the simplest procedure to find out the circular collision.

First, calculate the distance between the two origins of the circles:

$$((Ox2 - Ox1)^2 + (Oy2 - Oy1)^2)$$

sum of the
radius of the two circles:

```
if (originDistance <= ((R1 + R2) / 2))  
{  
    //Successfully Collide  
}  
else  
{  
    //Not Collide  
}
```

Performance comparison

For the first.

However, it is not that accurate. Particularly when developers work with a bigger circle, the lack in accuracy becomes visible.

. In the case
of many circles colliding in runtime, this process and mathematical calculation may cause performance delay.

The second approach is, overall, the worst possible way to solve this problem. However, this approach may be used in a very specific situation. When a developer is approach can be tried.

Detecting these sorts of collision may have multiple solutions. The approaches and solutions you have learned here are few of the most efficient solutions from the point of view of performance.

more popular
approach by creating a bigger round rectangle by increasing the width and height by

3D game development constraints

3D game development in Android native is very complicated. The Android framework does not support direct 3D game development platforms. 2D game development is directly supported by Android Canvas. The developer requires OpenGL support to develop 3D games for Android.

discuss a few constraints of 3D development for Android with OpenGL support.

Android provides the OpenGL library for development. The developer needs to set up scenes, light, and camera first to start any development process.

Vertices and triangles

Vertex refers to a point in 3D space. In Android, `Vector3` can be used to define the be projected onto a 2D plane. Any 3D object can be simplified to a collection of triangles surrounding its surface.

ube can be formed of 12 triangles as it has six surfaces. The number of triangles has a heavy impact on the rendering time.

3D transformation matrix

Each 3D object has its own transformation. `Vector` can be used to indicate its atrix called a transform matrix. A transformation matrix is 4 x 4 in dimension.

Let's assume the matrix to be T :

$$[T] = \begin{bmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \\ m & n & o & p \end{bmatrix}$$

Here:

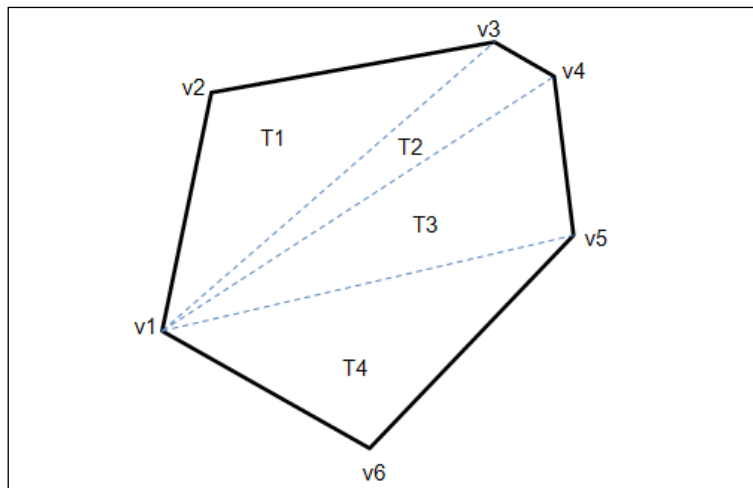
- $\{a, b, c, e, f, g, i, j, k\}$ represents linear transformation
- $\{d, h, l\}$ represents perspective transformation
- $\{m, n, o\}$ represents translations along the x , y , and z axes
- $\{a, f, k\}$ represents local scaling along the x , y , and z axes
- $\{p\}$ represents overall scaling
- $\{f, g, i, k\}$ represents rotation along the x axis where $a = 1$
- $\{a, c, i, k\}$ represents rotation along the y axis where $f = 1$
- $\{a, b, e, f\}$ represents rotation along the z axis where $k = 1$

form 3D
calculation.

As the number of vertices increases, the number of calculations increases as well.
This results in performance drop.

3D object and polygon count

Any 3D model or object has surfaces referred to as polygons. Fewer of polygons implies fewer of triangles, which directly decreases the vertices count:



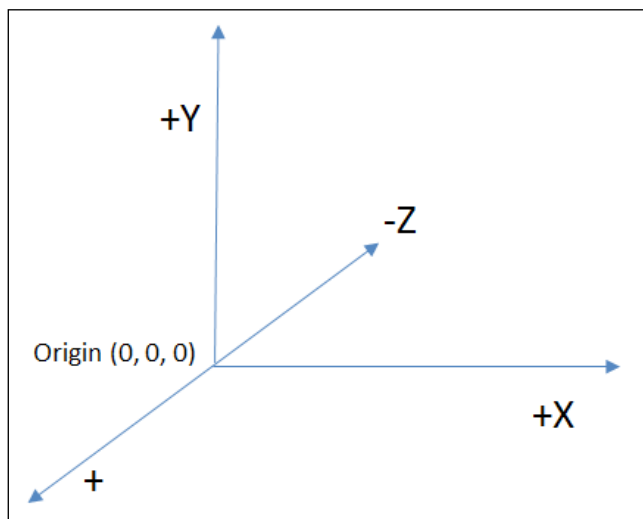
This is a simple example of a polygonal distribution of a 3D object surface. A six-sided polygon has four triangles and six vertices. Each vertex is a 3D vector. Every processor takes time to process each vertex. It is recommended that you keep a check on the total polygon count, which will be drawn in each draw cycle. Many games suffer a significant amount of FPS drop because of a high and unmanaged polygon count.

Android is specifically a mobile OS. Most of the time, it has limited device configuration a problem for developers.

3D rendering system

Android provides the `GLSurfaceView` and `GLSurfaceView.Renderer` to render 3D objects in Android. They are responsible for generating the model on screen. We have already discussed the 3D rendering pipeline through OpenGL.

3D rendering maps all the objects on a 3D world coordinate system following the right-hand thumb system:

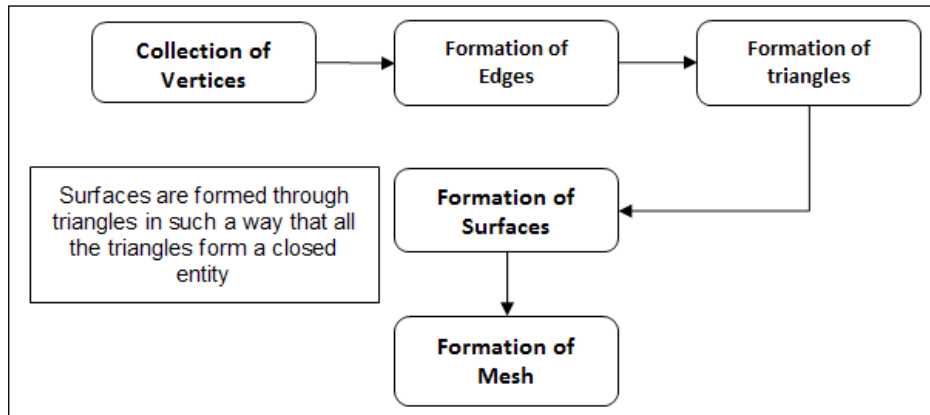


3D mesh

A 3D mesh is created with vertices, triangles, and surfaces. A mesh is created to represent the complete model.

imization can
be applied here.

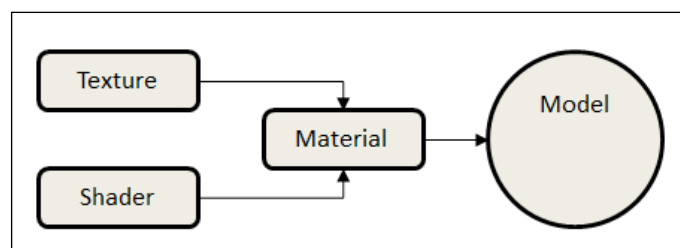
Here is the procedure of creating the mesh:



A 3D model can contain more than one mesh, and they may even be interchangeable. A mesh is responsible for the model detailing quality and for the rendering performance of the model. For Android development, it is recommended that you

Materials, shaders, and textures

is applied
on it to create the final model. However, the texture is applied through a material
and manipulated by shaders:



Textures

ew the
quality of a model. This image is mapped through the surfaces of the mesh so that
each surface renders a particular clip of the texture.

Shaders

Shaders are used to manipulate the quality, color, and other attributes of the texture. A texture with all the attributes properly set. A 3D model visibility is dependent on light source, intensity, color, and material type.

Materials

The material determines the texture attribute and shader property. The material can be termed as a container for the shader and texture before applying it to the mesh to create the model.

Collision detection

Collision detection for 3D Android games can be categorized into two types:

- Primitive colliders
- Mesh colliders

Primitive colliders

These colliders consist of basic 3D elements such as cubes, spheres, cylinders, prisms, and so on. This collision detection system follows certain geometric patterns and rules. That's why it is comparatively less complicated than the arbitrary mesh collider.

Most of the time, the developer assigns primitive colliders to many models to increase the performance of the game. This approach is obviously less accurate than actual collider.

Mesh colliders

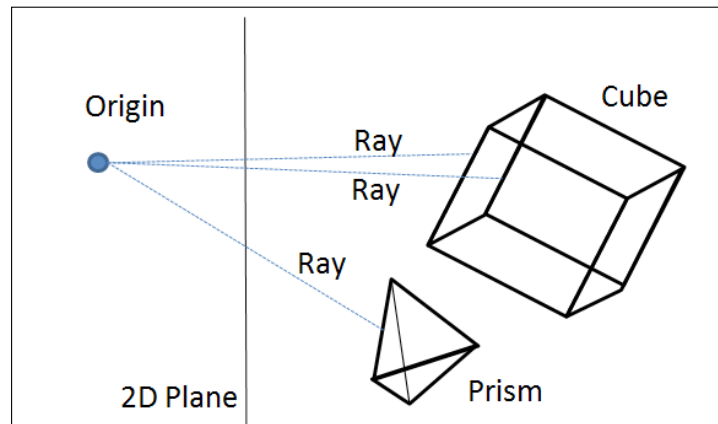
Mesh colliders can detect actual arbitrary collision detection. This collision detection reduces the process overhead. **quadtree**, **kd-tree**, and **AABB tree** are a few examples of such collision detection methods that reduce overhead significantly.

For each surface. To simplify this method, each mesh block is converted to boxes. A special AABB tree or quadtree is generated to reduce the vertex check.

This can be further reduced to **octree** vertex mapping by merging two box colliders. In this way, the developer can reduce the collision check to reduce CPU overhead.

Ray casting

objects. This system is used to solve the geometric problems of 3D computer graphics. In the case of 3D games, all 3D objects are projected in a 2D view. It is not possible to determine depth without ray casting in the case of a 2D electronic display:



hape of the
ing of the
objects, and so on.

In the case of Android games, ray casting is vastly used to handle touch input on the screen. Most of the games use this method to manipulate the behavior of 3D objects used in the game.

e costly
system to use in a major scale. This requires a series of geometrical calculation, resulting in a processing overhead. As the number of rays increases, the process gets heavier.

sting at one
point.

Concept of "world"

The word "world" in 3D games is a real-time simulation of the actual world with to actual objects in the real world. The scope of the game world is finite. This world follows a particular scale, position, and rotation with respective cameras.

The concept of camera is a must for simulating such a world. Multiple cameras can be used to render different perspectives of the same world.

s.
f the
parameters remain the same. These parameters are as follows:

- Finite elements
- Light source
- Camera

Elements of the game world

A world consists of the elements that are required in game design. Each game may n across he terrain, light sources at different scopes of the game.

Elements can be divided into two categories: movable objects and static objects. A game's rigid bodies are associated with such elements. Normally, static objects do not support motion physics.

t has a certain number of vertices and triangles. We have already discussed the processing basically the optimization of each element in the world.

Light sources in the game world

A game world must have one or more light sources. Lights are used to expose the on the user experience.

The game development process always requires at least one good light artist. shadow play in the game world is entirely dependent on light mapping.

however,
the consequence of processing light and shadow is a large amount of processing. All
ticular shader.

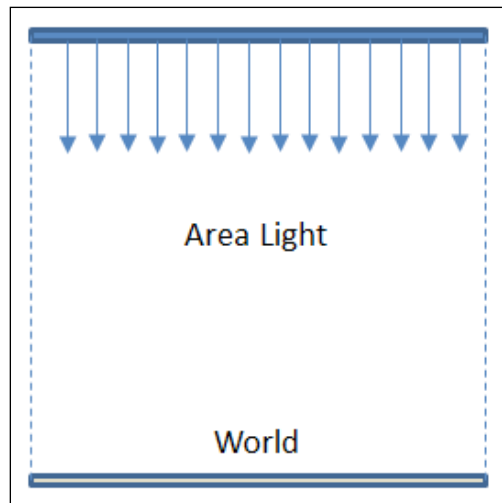
Use of extensive light sources results in low performance.

Light sources can be of the following types:

- Area light
- Spot light
- Point light
- Directional light
- Ambient light
- Volume light

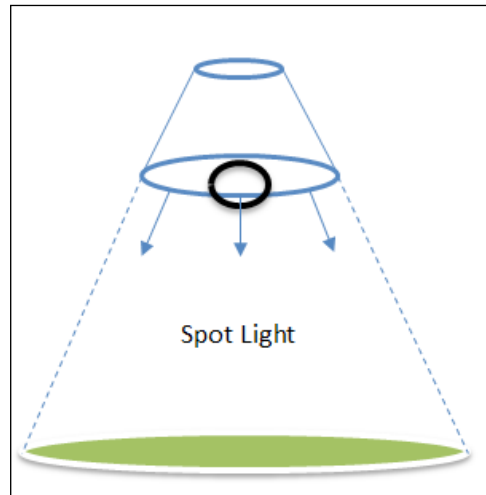
Area light

ion. By nature,
it is a directional light and lights the area with equal intensity:



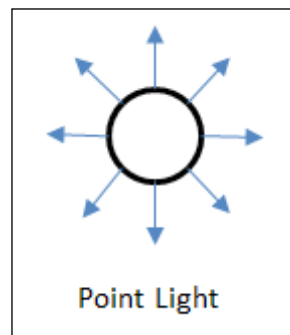
Spot light

A spot light is used to focus on a particular object in a conical directional shape:



Point light

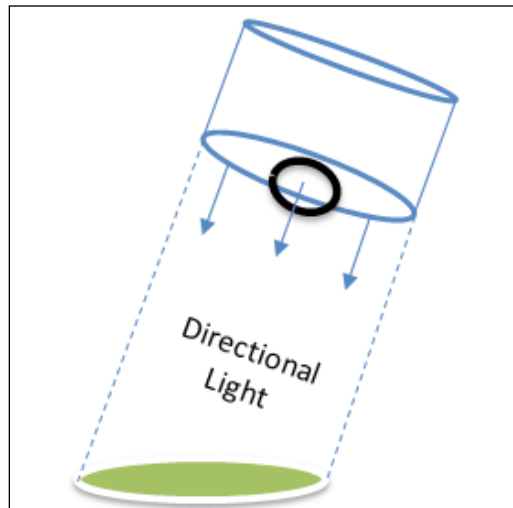
Example is a bulb
illumination:



Directional light

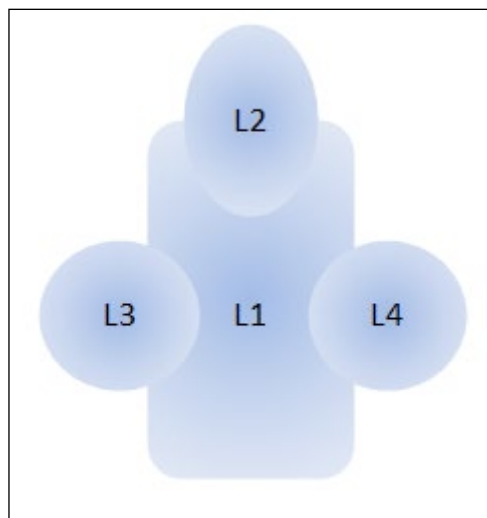
e in a 3D world.

A typical example is sunlight:



Ambient light

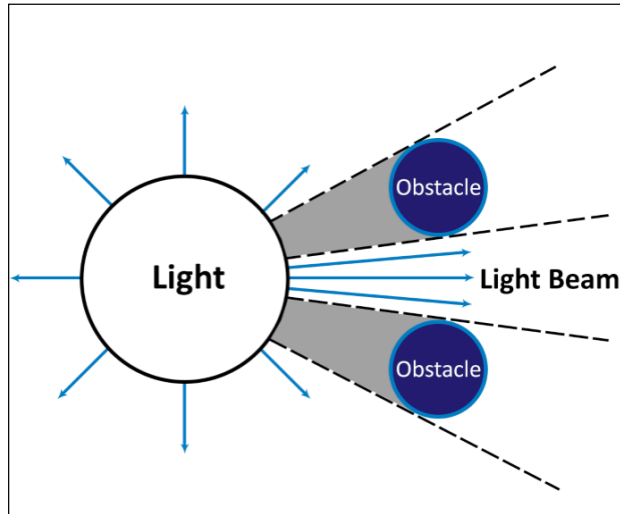
An ambient light is a set of arbitrary light beams in any direction. Usually, the follow a particular direction, and it does not generate any shadows:



L1, **L2**, **L3**, and **L4** are ambient light sources here.

Volume light

A volume light is a modified type of point light. This kind of light source can be converted into a set of light beams within a defined geometrical shape. Any light beam is a perfect example of such a light source:



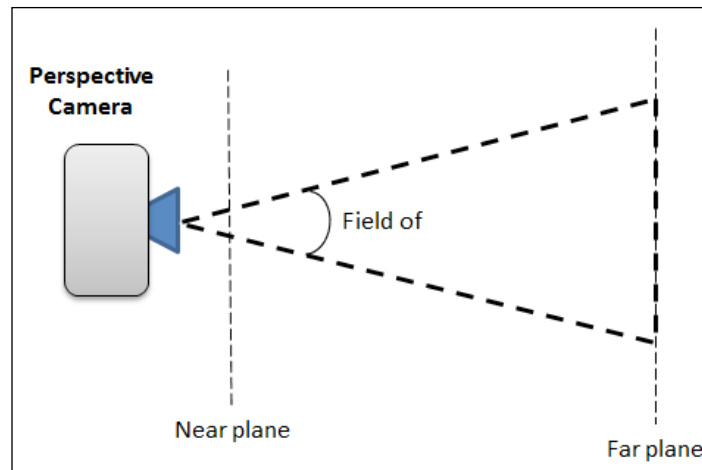
Cameras in the game world

A camera
the elements to
be added in the rendering pipeline.

There are two types of camera used in a game.

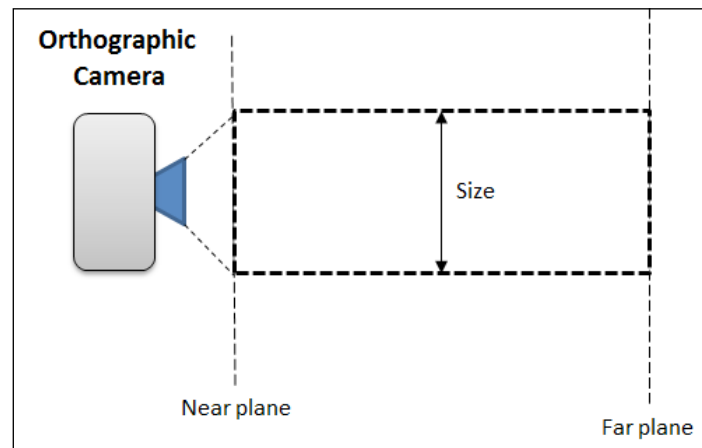
Perspective camera

This type of camera is typically used to render 3D objects. The visible scale and depth field of view and near/far range to control the rendering pipeline:



Orthographic camera

This type of camera is used to render objects from a 2D perspective, irrespective of perspective. The camera to games and to render 2D objects in a 3D game:



Besides this, the game camera can also be categorized by their nature and purpose. Here are the most common variations.

Fixed camera

A fixed camera is used in many games. A fixed camera is the most convenient camera in terms of processing speed. A fixed camera does not have any runtime manipulation.

Rotating camera

This camera has a rotating feature during runtime. This type of camera is effective in the case of sports simulation or surveillance simulation games.

Moving camera

A camera can be said to be moving when the translation can be changed during game. A typical use of this sort of camera is for games such as *Age Of Empires*, *Company Of Heroes*, *Clash Of Clans*, and so on.

Third-person camera

ra, but this camera follows a particular object or character. The character is supposed to be the user character, so all the actions and movements are tracked by this camera including according to the actions of the player.

First-person camera

When the player plays as the main character, this camera is used to implement a according to the actions of the player.

The rendering pipeline in Android

Let's now have a look at the types of rendering pipeline in Android.

The 2D rendering pipeline

In the case of the 2D Android drawing system through Canvas, all the assets are first drawn on the canvas, and the canvas is rendered on screen. The graphic engine maps all the assets within the finite Canvas according to the given position.

Often, developers use small assets separately that cause a mapping instruction to execute for each asset. It is always recommended that you use sprite sheets to merge as many small assets as possible. A single draw call can then be applied to draw every object on the Canvas.

ences are.

Previously, Android could not support images or sprites of a size more than 1024 x 1024 pixels. However,

using such sprites can cause permanent memory occupancy during the scopes of all the small assets. Many low-configuration Android devices do not support such large textures. Developers limit

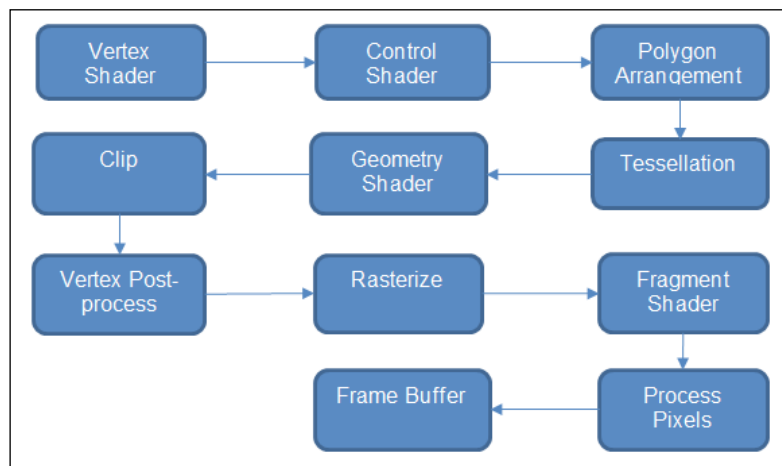
themselves to 2048 x 2048 pixels. This will reduce memory usage peak, as well as significant amounts of draw calls to the canvas.

The 3D rendering pipeline

pipeline for

Android 3D is basically the OpenGL pipeline.

Let's have look at the OpenGL rendering system:



Now, let's have a detailed look at each step of the preceding rendering flow diagram:

1. The **vertex shader** processes individual vertices with vertex data.
2. The **control shader** is responsible for controlling vertex data and patches for the tessellation.

3. The **polygon arrangement** system arranges the polygon with each pair of
ut
repeating vertices.
4. **Tessellation** is the process of tiling the polygons in a shape without overlap
or any gaps.
5. The **geometry shader** is responsible for optimizing the primitive shape. Thus
triangles are generated.
6. After constructing the polygons and shapes, the model is **clipped** for
optimization.
7. **Vertex post processing** is used to filter out unnecessary data.
8. The mesh is then **rasterized**.
9. The **fragment shader** is used to process fragments generated from
rasterization.
10. All the pixels are **mapped** after fragmentation and **processed** with the
processed data.
11. The mesh is added to the **frame buffer** for final rendering.

Optimizing 2D assets

assets in
some form inside the game. So, as far as game component optimization is concerned,
every 2D asset should also be optimized. Optimization of 2D assets means these
three main things.

Size optimization

Each asset frame should only contain the effective pixels to be used in games.
Unnecessary pixels increase the asset size and memory use during runtime.

Data optimization

Not all images require full data information for pixels. A significant amount of data
e, full
screen opaque images should never contain transparency data. Similarly, depending
ormat.

Image optimization tools can be used to perform such optimizations.

Process optimization

time it

takes to decompress it and load it to memory. So, image optimization has a direct effect on the processing speed.

another way to

reduce the processing time of images.

Optimizing 3D assets

optimized

consider is

ure.

vertices and

the model polygon.

Limiting the polygon count

can create

more details. However, we all know that Android is a mobile OS, and it always has hardware limitations.

the total

is always a

limitation depending on the hardware configuration.

e in order

to achieve a certain frame rate or performance.

Model optimization

Models are created with more than one mesh. Using a separate mesh in the final model always results in heavy processing. This is a major effort for the game artist. Multiple overlaps can occur if multiple meshes are used. This increases vertex processing.

Rigging is another essential part of finalizing the model. A good rigger defines the skeleton with the minimum possible joints for minimum processing.

Common game development mistakes

at every development stage. It is a very common practice to use assets and write code in a temporary mode and use it in the final game.

There are few of the most common mistakes made during game development.

Use of non-optimized images

are loaded into the game unoptimized, even for the release candidate.

Images that contain limited information. Alpha information may be found in opaque images.

Use of full utility third-party libraries

Each module should be written from scratch. Most of the developers use a predefined third-party library for common utility mechanisms.

Most of the time, these packages come with most of the possible methods, and over time, use these packages without any filtration. A lot of unused data occupies memory during runtime in such cases.

In each case, the developer should choose such packages very carefully, depending on their specific requirements.

Use of unmanaged networking connections

In modern Android games, the use of Internet connectivity is very common. Many games use server-based gameplay. In such cases, the entire game runs on the server with frequent data transfers between the server and the client device. Each data transfer process takes time, and the connectivity drains the battery charge significantly.

Badly managed networking states often freeze the application. A significant amount of time, in some cases, is spent on a request.

The developer often skips this part to save development time.

Another aspect of unmanaged connections is unnecessary packet data transferred over each time data is transferred.

Using substandard programming

We have already discussed programming styles and standards. The modular

management of programming demands modular programming. Otherwise, developers end up repeating code, and this increases process overhead.

Memory management also demands a good programming style. In few cases, the use of memory leads to memory leakage. At times, the application crashes due to insufficient memory.

Substandard programming includes the following mistakes:

- Declaring the same variables multiple times
- Creating many static instances
- Writing non-modular coding
- Improper singleton class creation
- Loading objects at runtime

Taking a shortcut

a shortcut during development is very common among game developers.

ways
convenient
bubble
that it is the
most inefficient sorting process.

Using such shortcuts multiple times in a game may cause a visible process delay, which directly affects the frame rate.

2D/3D performance comparison

game
he
deciding factor.

Different look and feel

in 3D games is
very common to provide visual effects. In the case of 2D games, sprite animation and other transformations are used to show such effects.

shadow.
ost 3D
games use dynamic lighting, which has a significant effect on game performance. In
is no extra
processing in 2D games for light and shadow.

In 2D games, the game screen is rendered on a Canvas. There is only one fixed point of view. So, the concept of camera is limited to a fixed camera. However, in 3D
ed. Multiple
jects through
multiple cameras causes more process overhead. Hence, it decreases the frame rate of the game.

There is a significant performance difference between using 2D physics and 3D physics. A 3D physics engine is far more process heavy than a 2D physics engine.

3D processing is way heavier than 2D processing

games in
comparison to 2D games. In Android, the standard accepted FPS for 2D games is as 40 FPS.

games in terms of process. The main reasons are as follows:

- **Vertex processing:** In 3D games, each vertex is processed on the OpenGL eavier processing.
- **Mesh rendering:** A mesh consists of multiple vertices and many polygons. Processing a mesh increases the rendering overhead as well.
- **3D collision system:** A 3D dynamic collision detection system demands each vertex of the collider to be calculated for collision. This calculation is usually done by the GPU.
- **3D physics implementation:** 3D transformation calculation completely depends on matrix manipulation, which is always heavy.
- **Multiple camera use:** Use of multiple cameras and dynamically setting up the rendering pipeline takes more memory and clock cycles.

Device configuration

Android has a wide range of device configuration options supported by the . Running the same game on different configurations does not produce the same result.

Performance depends on the following factors.

Processor

There are many processors used for Android devices in terms of the number of cores and the speed of each core. Speed decides the number of instructions that can be executed in a single cycle. There was a time when Android used to have a single core CPU with speed less than 500 MHz. Now we have multicore CPUs with more than 2 GHz speed on each core.

RAM

mes frequent loading/unloading processes affect performance.

GPU

GPU decides the rendering speed. It acts as the processing unit for graphical objects. A more powerful processor can process more rendering instructions, resulting in better performance.

Display quality

Display quality is actually inversely proportional to the performance. Better display quality has to be backed by better GPU, CPU, and RAM, because better displays cost more.

We can see various devices with different display quality. Android itself has divided the assets by this feature:

- **LDPI:** Lowest dpi display for Android (~120 dpi)
- **MDPI:** Medium dpi display for Android (~160 dpi)
- **HDPI:** High dpi display for Android (~240 dpi)
- **XHDPI:** Extra high dpi display for Android (~320 dpi)
- **XXHDPI:** Extra extra high dpi display for Android (~480 dpi)
- **XXXHDPI:** Extra extra extra high dpi display for Android (~640 dpi)

in the near future,
with the advancement of hardware technology.

Battery capacity

More powerful CPUs, GPUs, and RAM demand more power. If the battery is incapable of delivering power, then processing units cannot run at their peak efficiency.

To summarize these factors, we can easily make a few relational equations with performance:

- CPU is directly proportional to performance
- GPU is directly proportional to performance
- RAM is directly proportional to performance
- Display quality is inversely proportional to performance
- Battery capacity is directly proportional to performance

Summary

The scope of 3D games is increasing day by day with more quality and performance. . Old devices are not obsolete yet.

It becomes a serious problem when the same application runs on various devices. This becomes a challenge for developers to run the same application across devices.

rendering, processing, and assets. The developer should always use an optimized mance is to port the games for different hardware systems for both 2D and 3D games.

decade.

Accordingly, the nature of games has also changed. However, the scope of 2D games is still there with a large set of possibilities.

games.
both 2D and
3D games.

Improving performance is more of a logical task than a technical one. There are a choose them.

So, selecting the right tool for the right purpose is necessary, and there should be a different approach to making 2D and 3D games.

We have already discussed the rendering processes in both 2D and 3D development. d try to k.

7

Working with Shaders

Every game's success depends largely on its look and feel. This directly means that possible actions. runtime for display. This necessity gave birth to the concept of shaders.

rawable elements before rendering. Mostly, shaders are optimized for a specific graphics processor. However, nowadays, shaders can be written to support multiple processors on multiple platforms.

Android accommodates the option to work with shaders in the Android framework itself. Additionally, OpenGL shaders can also be used and customized with the help of the Android NDK. There are many occasions where exquisite graphical quality is delivered with the help of shaders without excellent raw art assets.

We will have a discussion about shaders in this chapter from the point of view of Android game development through the following topics:

- Introduction to shaders
- How shaders work
- Types of shaders
- Android library shaders
- Writing a custom shader
- Shaders through OpenGL
- Use of shaders in games
- Shaders and game performance

Introduction to shaders

Many developers develop games on Android, but do not possess much knowledge about shaders. In most cases, developers do not need to work with shaders, or there are some pre-defined shaders inside the game development framework or engines.

In 1988, the animation studio Pixar introduced the modern concept of shaders.

. OpenGL

and Direct3D are the first two graphic libraries to support shaders. GPU started enhanced

by

OpenGL 3.2 and Direct3D 10 libraries.

Let's now dive a bit deeper into shaders to understand their definition, necessity, and scope for Android games.

What is a shader?

In simple words, a shader is an instruction set to manipulate the visual display of the input graphic assets.

Let's elaborate the definition a bit. All the instructions are basically done through computer

tion and

input asset to produce more efficient output-displayable assets.

Typical shaders can process either a vertex or a pixel. Pixel shaders can compute n compute

the position, color, co-ordinates, depth, illuminations, and so on of a vertex. Thus, erational

base type:

- 2D shaders
- 3D shaders

Necessity of shaders

In the normal practice of game development, Android developers do not bother r small-scale

games, art assets are used without improvement. Any modification to the assets is managed by the old process of updating the art asset itself.

Shaders can minimize this extra time-consuming effort. The same asset can be blurred out during gameplay to indicate different teams or players, create masks of art assets, and so on.

Shaders have the following benefits:

- When different shaders are applied to the same art asset, it produces different assets, depending on your requirements at runtime. Thus, the shader can save extra art-creation time.
- One-time integration of drawable objects in the game can lead to a different visual experience through different shaders.
- As the art assets are minimized, using shaders can reduce the game build size.
- There will be more visual difference with the same set of assets.
- Animation can be created by shaders with simple art by manipulating the visual content repeatedly.
- Shaders are useful for creating visual effects during runtime.

However, using shaders may lead to some negative consequences:

- Using shaders will increase the processing time due to the manipulation of the visual assets during runtime
- Unoptimized use of shaders may lead to more heap memory consumption as various intermediate instances will be stored in it
- Sometimes, shaders are responsible for the distortion of objects while processing
- Art assets become vulnerable to quality loss using shaders

Only the first two are actual direct consequences of using shaders. The rest of the consequences are indirect or faulty. Therefore, it is extremely necessary to choose the perfect shader for a specific task.

Sometimes, the shader process takes a long time, resulting in poor FPS output. A few old GPUs do not support all kinds of shaders. Therefore, the developer should check and confirm the hardware platform on which the shader is to perform.

Scope of shaders

, such as
games,
and so on.

The gaming sector is one of the largest communities that uses shaders. The Android platform is no exception. Android game developers use shaders on a large scale in both 3D and 2D games.

xel
. This is
useful when the same raw assets are used for different visibility.

For example, a 2D cricket game has many teams with different outfits to distinguish the design.

Thus, the
output sprites have different visibility and are recognized easily by the player.

How shaders work

. So, the basic
working principle is to change or manipulate data at runtime:



A shader process is a specific set of instructions to process vertices or fragments.
.

A vertex shader is used to change the shape of the model; it can also change the surface-formation system.

ty. Pixel
data can be merged, modified, or replaced by a shader program to form a new digital image.

Types of shaders

on the basis of their behavior and features. Some of the shaders are as follows:

- Pixel shaders
- Vertex shaders
- Geometry shaders
- Tessellation shaders

Let's have a detailed look at these types.

Pixel shaders

Pixel shaders process colors and other attributes of a single pixel. Each single pixel is called a pixel shader.

Vertex shaders

A vertex shader mainly operates on the vertices of a mesh or model. Every mesh of a model is made up of multiple vertices. A vertex shader can only be applied to 3D models. So, a vertex shader is a type of 3D shader.

Geometry shaders

Geometry shaders are used for applying geometry to create points, lines, and triangles to form a surface.

Tessellation shaders

This is a typical 3D shader used to simplify and improve 3D mesh during tessellation. It is subdivided into two shaders:

- Hull shaders or tessellation control shaders
- Domain shaders or tessellation evaluation shaders

These two shaders are used together to reduce mesh bandwidth.

Tessellation shaders have the power to improve 3D models in such a way that the drawable vertex count is reduced significantly. Thus, rendering becomes faster.

Android library shaders

Android provides the shader option in its framework in the `android.graphics` package. A few well-known and widely used shaders are also in the Android library. Some of them are as follows:

- `BitmapShader`: This can be used to draw a bitmap in the texture format. It reating terrain with tiling.
- `ComposeShader`: This is used to merge two shaders. So, it is very useful for masking or merging colors for two different shaders.
- `LinearGradient`: This is used to create a gradient along with the given line segment with a defined color set.
- `RadialGradient`: This is used to create a gradient along with the given circle segment with a defined color set. A radial origin and radius are provided to create the gradient.
- `SweepGradient`: This is used to create a sweep gradient color around a point with the given radius.

Here is an example:

```
@Override
protected void onDraw ( Canvas c )
{
    float px = 100.0f;
    float py = 100.0f;
    float radius = 50.0f;

    int startColor = Color.GREEN;
    int targetColor = Color.RED;

    Paint shaderPaint = new Paint();
    shaderPaint.setStyle(Paint.Style.FILL);

    //LinearGradient Example to a circular region
    LinearGradient lgs = new LinearGradient( px, py, px + radius, py +
radius,
        startColor, targetColor, Shader.TileMode.MIRROR);
    shaderPaint.setShader(lgs);
    c.drawCircle( px, py, radius, shaderPaint);

    //RadialGradient Example to a circular region
    px = 200.0f;
```

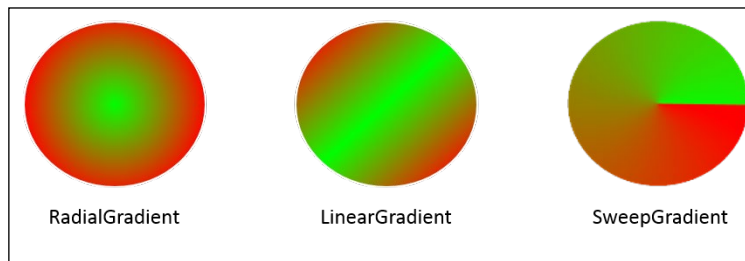
```

py = 200.0f;
RadialGradient rgs = new LinearGradient( px, py, radius,
    startColor, targetColor, Shader.TileMode.MIRROR);
shaderPaint.setShader(rgs);
c.drawCircle( px, py, radius, shaderPaint);

//SweepGradient Example to a circular region
px = 300.0f;
py = 300.0f;
shaderPaint.setShader(new SweepGradient(px, py, startColor,
targetColor));
c.drawCircle( px, py, radius, shaderPaint);
}

```

Here is what it looks like:



ent styles of the
same primitive object.

Writing custom shaders

uirements.

Android provides the `android.graphics.Shader` class. It is easy to create your own shader class using the primitive shaders provided.

The custom shader may not include only one shader. It can be a combination of various shaders. For example, consider masking an image with a circular view port with a motion-touch event:

```

private float touchX;
private float touchY;

```

```
private boolean shouldMask = false;

private final float viewRadius;
private Paint customPaint;

@Override
public boolean onTouchEvent(MotionEvent motionEvent)
{
    int pointerAction = motionEvent.getAction();
    if ( pointerAction == MotionEvent.ACTION_DOWN ||
        pointerAction == MotionEvent.ACTION_MOVE )
        shouldMask = true;
    else
        shouldMask = false;

    touchX = motionEvent.getX();
    touchY = motionEvent.getY();
    invalidate();
    return true;
}

@Override
protected void onDraw(Canvas canvas)
{
    if (customPaint == null)
    {
        Bitmap source = Bitmap.createBitmap( getWidth(), getHeight(),
            Bitmap.Config.ARGB_8888);
        Canvas baseCanvas = new Canvas(source);
        super.onDraw(baseCanvas);

        Shader customShader = new BitmapShader(source,
            Shader.TileMode.CLAMP, Shader.TileMode.CLAMP);

        customPaint = new Paint();
        customPaint.setShader(customShader);
    }

    canvas.drawColor(Color.RED);
    if (shouldMask)
    {
        canvas.drawCircle( touchX, touchY - viewRadius, viewRadius,
            customPaint);
    }
}
```

This example is one of the most commonly used shader styles in picture-based games. You can also implement such shaders to create hidden object games.

Another use case is highlighting a specific object on the screen. The same viewable color can be semitransparent to show a dull background.

Shaders through OpenGL

In Android, OpenGL supports implementing shaders for Android 3D games. In two functional segments while manually creating the shader:

- Shader
- Program

The shader is converted into intermediate code to support the program to run on the device. Shaders need to be recompiled before the program execution.

We will work with `GLSurfaceView` of the `android.opengl` package in our example.

For 3D games, an Android developer can use this package to play with shaders on the Android SDK. This package provides the API to create and use an OpenGL shader with Java.

We will use `GLSurfaceView` instead of the normal Android `View` or `SurfaceView`. The implementation will look like this:

```
import android.opengl.GLSurfaceView;
import android.content.Context;

public class MyGLExampleView extends GLSurfaceView
{
    private final GLRenderer mRenderer;

    public MyGLExampleView (Context context)
    {
        super(context);

        // Set OpenGL version 2.0 as we will be working with that particular
        // library
        this.setEGLContextClientVersion(2);

        // Set the Renderer for drawing on the GLSurfaceView
    }
}
```



```
        MyOpenGLRendererExample = new MyOpenGLRendererExample
            (context);
        setRenderer(mRenderer);

// Render the view only when there is a change in the //drawing data
        setRenderMode(GLSurfaceView.RENDERMODE_CONTINUOUSLY);
    }

    @Override
    public void onPause()
    {
        super.onPause();
    }

    @Override
    public void onResume()
    {
        super.onResume();
    }
}

:

import java.nio.ByteBuffer;
import java.nio.ByteOrder;
import java.nio.FloatBuffer;
import java.nio.ShortBuffer;

import android.content.Context;
import android.opengl.GLES20;
import android.opengl.GLSurfaceView.Renderer;
import android.opengl.Matrix;
import android.util.Log;

import javax.microedition.khronos.egl.EGLConfig;
import javax.microedition.khronos.opengles.GL10;

public class MyOpenGLRendererExample implements Renderer
{

    // Declare matrices
    private float[] matatrixProjection = new float[32];
    private float[] matrixView = new float[32];
    private float[] matatrixProjectionOnView = new float[32];
```

```
// Declare Co-ordinate attributes
private float vertexList[];
private short indicxList[];
private FloatBuffer vertexBuffer;
private ShortBuffer drawBuffer;

private final String vertexShader =
    "uniform    mat4        uMVPMatrix;" +
    "attribute  vec4        vPosition;" +
    "void main() {" +
    "    gl_Position = uMVPMatrix * vPosition;" +
    "}";

private final String pixelShader =
    "precision mediump float;" +
    "void main() {" +
    "    gl_FragColor = vec4(0.5,0,0,1);" +
    "}";

// Declare Screen Width and Height HD display
float ScreenWidth = 1280.0f;
float ScreenHeight = 800.0f;

private int programIndex = 1;

public MyOpenGLRendererExample (Context context)
{

}

@Override
public void onDrawFrame(GL10 param)
{
    renderView(matatrixProjectionOnView);
}

@Override
public void onSurfaceChanged(GL10 objGL, int width,
int height)
{
    ScreenWidth = (float)width;
    ScreenHeight = (float)height;

    GLES20.glViewport(0, 0, (int)ScreenWidth,
```

```
(int)ScreenHeight);

    //reset matrices
    for( int i = 0; i < 32 ; ++ i )
    {
        matatrixProjection[i] = 0.0f;
        matrixView[i] = 0.0f;
        matatrixProjectionOnView[i] = 0.0f;
    }

    Matrix.orthoM(matatrixProjection, 0, 0f, ScreenWidth,
0.0f, ScreenHeight, 0, 50);

    Matrix.setLookAtM(matrixView, 0, 0f, 0f, 1f, 0f, 0f,
0f, 0f, 1.0f, 0.0f);

    Matrix.multiplyMM(matatrixProjectionOnView, 0,
matatrixProjection, 0, matrixView, 0);
}

@Override
public void onSurfaceCreated(GL10 gl, EGLConfig config)
{
    //create any object
    //Eg. Triangle:: simplest possible closed region

    createTriangle();

    // Set the color to black
    GLES20.glClearColor(0.0f, 0.0f, 0.0f, 1);

    // Create the shaders
    int vertexShaderTmp =
loadShader(GLES20.GL_VERTEX_SHADER, vertexShader);

    int pixelShaderTmp =
loadShader(GLES20.GL_FRAGMENT_SHADER, pixelShader);

    int programIndexTmp = GLES20.glCreateProgram();

    GLES20.glAttachShader(programIndexTmp,
vertexShaderTmp);

    GLES20.glAttachShader(programIndexTmp,
```

```
pixelShaderTmp);

    GLES20.glLinkProgram(programIndexTmp);

    // Set shader program
    GLES20.glUseProgram(programIndexTmp);
}

void renderView(float[] matrixParam)
{
    int positionHandler =
    GLES20.glGetAttribLocation(programIndex, "vPosition");

    GLES20.glEnableVertexAttribArray(positionHandler);
    GLES20.glVertexAttribPointer(positionHandler, 3,
    GLES20.GL_FLOAT, false, 0, vertexBuffer);

    int mtrxhandle =
    GLES20.glGetUniformLocation(programIndex, "uMVPMatrix");

    GLES20.glUniformMatrix4fv(mtrxhandle, 1,
    false, matrixParam, 0);

    GLES20.glDrawElements(GLES20.GL_TRIANGLES,
    indicxList.length, GLES20.GL_UNSIGNED_SHORT, drawBuffer);

    GLES20.glDisableVertexAttribArray(positionHandler);
}

void createTriangle()
{
    // We have to create the vertexList of our triangle.
    vertexList = new float[]
    {
        20.0f, 200f, 0.0f,
        20.0f, 300f, 0.0f,
        200f, 150f, 0.0f,
    };

    //setting up the vertex list in order
    indicxList = new short[] {0, 1, 2};

    ByteBuffer bytebufVertex =
```

```
ByteBuffer.allocateDirect(vertexList.length * 4);

bytebufVertex.order(ByteOrder.nativeOrder());
vertexBuffer = bytebufVertex.asFloatBuffer();
vertexBuffer.put(vertexList);
vertexBuffer.position(0);

ByteBuffer bytebufindex =
ByteBuffer.allocateDirect(indicxList.length * 2);

bytebufindex.order(ByteOrder.nativeOrder());
drawBuffer = bytebufindex.asShortBuffer();
drawBuffer.put(indicxList);
drawBuffer.position(0);

int vertexShaderTmp =
loadShader(GLES20.GL_VERTEX_SHADER, vertexShader);

int pixelShaderTmp =
loadShader(GLES20.GL_FRAGMENT_SHADER, pixelShader);

int program = GLES20.glCreateProgram();
if (program != 0)
{
    GLES20.glAttachShader(program, vertexShaderTmp);
    GLES20.glAttachShader(program, pixelShaderTmp);
    GLES20.glLinkProgram(program);

    int[] linkStatus = new int[1];

    GLES20.glGetProgramiv(program,
    GLES20.GL_LINK_STATUS, linkStatus, 0);

    if (linkStatus[0] != GLES20.GL_TRUE)
    {
        Log.e("TAG_EXAMPLE_OPENGL", "Linking Failed !!
Error:: " + GLES20.glGetProgramInfoLog(program));

        GLES20.glDeleteProgram(program);
        program = 0;
    }
}
}
```

```
// method to create shader
int loadShader(int type, String shaderCode)
{
    int shader = GLES20.glCreateShader(type);

    GLES20.glShaderSource(shader, shaderCode);
    GLES20.glCompileShader(shader);

    return shader;
}
```

The vertex shader code (String `vs_SolidColor`) has two parameters that it needs. The `uMVPMatrix` parameter is of the type `mat4`, which holds in the transformation matrix that can be used to translate the position. The `uMVPMatrix` parameter is a uniform matrix. The `vPosition` parameter is of type `vec4`, which holds the positions of vertex.

This system can be applied for a triangular surface.

Use of shaders in games

dynamic lighting, changing tints, and making dynamic visual improvements. Sometimes, the world environment is created with shaders.

Shaders in a 2D game space

e is considered a fragment. This is the reason why pixel shaders are also called fragment shaders. Pixel shaders can only perform color changes, tiling, and masking.

`BitmapShader`, `ComposeShader`, `LinearGradient`, `RadialGradient`, and `SweepGradient` are the variants of Android 2D shaders.

A 2D game world is created with images. Developers often choose to create different assets to give the same object a different look and feel. In this process, developers end up making a bigger APK with almost the same use set.

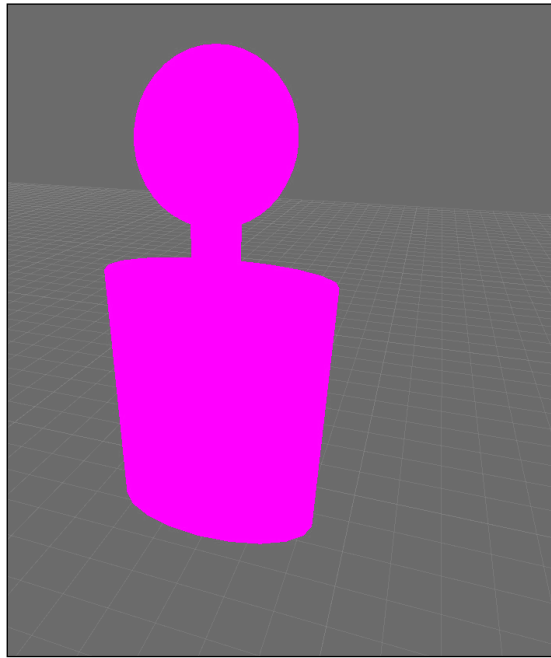
Sprites can also be a field where shaders can hold a significant role. When using the need to change dynamically. Pixel shaders can be very useful here.

Shaders in a 2D space are used to change color, blur segments, change brightness, change opacity, tint images, and so on.

Shaders in a 3D game space

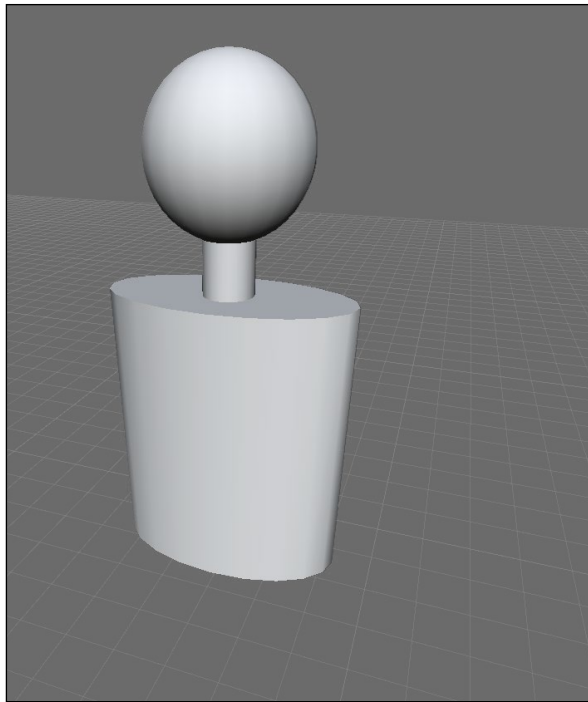
The most common use of shaders in 3D games is for dynamic shadow. In modern experience, 3D models look real after applying a texture.

In Android, a 3D shader is applied through OpenGL. We have already discussed an example:



A raw model with only vertex information

This is a simple model without any lightening or shaders. Let's apply some shaders to give it a solid 3D look:

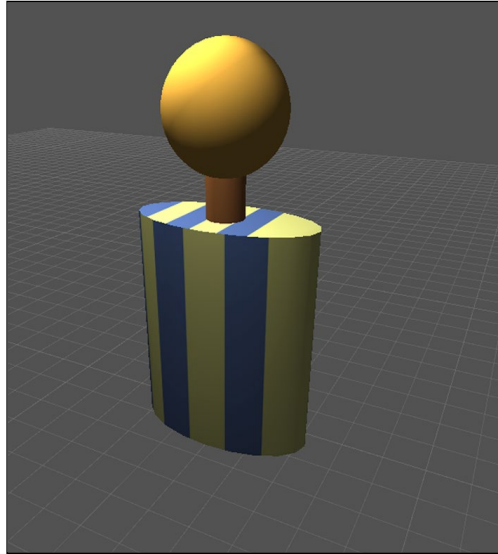


A simple flat shader applied

feel. In
re. Generally,
usually real.
However, this costs more than just color manipulation.

nt feel of the
same object. There are different procedures to handle different scenario requirements
for the game.

This example, however, is just a visual representation of how shaders can manipulate 3D models for a different look and feel:



Shaders and game performance

Shaders are usually process-heavy. A fragment shader processes each fragment of a texture and manipulates its data. A large texture may lead to a visible delay in the game loop.

We can see shaders from different perspectives to create an idea of performance.

Real-time use
of shaders.

Creating shadows is one of the extensive uses of shaders. However, the quality of shadow processing is inversely proportionate to performance. In high-quality games, we can experience real-time shadow. Shaders map the object vertices and process it according to the light direction. It is then projected on the X-Z plane to create shadow. Shadows are merged with objects on the plane and with other shadows.

Shaders can be used to improve world visibility with different lights, materials, and colors.

Here are some pros of using shaders in games:

- Complete flexibility when rendering assets
- Fewer asset packages and increased reusability
- Dynamic visual effects
- Dynamic lighting and shadow
- Sprite manipulation on the fly

There are few disadvantages of using shaders:

- Comparatively low frame rate
- Performance drop
- Required supported hardware platforms and graphic drivers

intrinsic part of game development. Any performance drop is handled by upgrading the hardware and graphic drivers.

Nowadays, shaders are being optimized for embedded devices with limited resources. This even opens up the chance to increase the use of shaders on almost every platform, without affecting the performance significantly.

Summary

Since Android API level 15, the framework supports OpenGL ES 2.0. This gave immense flexibility to graphic programmers to implement shaders in Android games.

Almost every hardware configuration supports shaders to run on GPU. However, the scale of using shaders determines the performance. In modern day, this is not actually an issue.

programming, shaders have already proven their place. All the famous and successful game developers have acknowledged the importance of shaders. Graphic artists need not worry about everything visual in the game, which reduces the development time significantly.

Shaders are, therefore, widely used in games. Newer shaders are coming up with additional features now. The upgrading cycle of shaders has become less. However, physical updates.

It feels like magic to see a simple cube turn into anything that has the same ure.

ory usage,
timization
techniques of storage and processing in the next chapter.

8

Performance and Memory Optimization

. It
nce
significantly. Through optimization, more hardware platforms can be targeted.

orms.

Each platform has a separate configuration. By optimizing the use of hardware resources, a game can be run on more hardware platforms. This technique can be display, so
optimizing the assets for low resolution saves a lot of storage space as well as heap memory during runtime.

to
optimize it later. This may cause a significant amount of performance loss or even cause the game to crash.

We will discuss the scope of various optimizations in Android game development through the following topics:

- Fields of optimization in Android games
- Relationship between performance and memory management
- Memory management in Android
- Processing segments in Android
- Different memory segments
- Importance of memory optimization
- Optimizing performance
- Increasing the frame rate

- Importance of performance optimization
- Common optimization mistakes
- Best optimization practices

Fields of optimization in Android games

In the case of game development, this fact remains the same. In a game development project, the process starts with limited resources and design. After development, the game is achieved that, memory and performance optimization becomes mandatory. So, let's discuss the following four segments of optimization:

- Resource optimization
- Design optimization
- Memory optimization
- Performance optimization

Resource optimization

Resource optimization is basically optimizing the art, sound, and data files.

Art optimization

We have already discussed many optimization techniques and tools. Here, we will discuss the necessity of art optimization.

Art is visually the most important part in games. Improving the art with bigger and better display quality increases processing and storage costs.

to fit

met. Also,

various Android devices support various limitations on texture size. Moreover, it takes more time for a shader to work on a larger texture.

completely opaque texture. This data increases the texture size significantly.

Art assets can be optimized on the art style. Many developers use flat-colored texture pixel data.

This again saves disk space and processing time.

f them to
increase flexibility in order to create quality visual art without spending much time on optimization.

Sound optimization

Sound is another vital resource for games. Audio may be compressed to save space and effort. A common practice in the Android game industry is to use a compressed format for long audio files.

It takes time to compress and decompress files during runtime. So, using SFX dynamically can be a problem if it is compressed. It can trigger a significant and visible stutter. Developers like to use an uncompressed format for SFX and a compressed format for long and continuous playing sounds such as background music.

Data file optimization

Sometimes, game developers use separate data files to create a flexible project structure to interact with external tools or for better data interface. Such files are their own data format in a binary model.

here is not
to keep a
check on the amount of data and the total file size.

Design optimization

Design optimization is used to increase the scalability, quality experience, flexibility, the game parameters around the core game concept.

ionality:

- Game design optimization
- Technical design optimization

Game design optimization

design
The
developer needs to find different ways to communicate the basic game idea. Then, they can choose the best one, following some analysis.

Game design should be flexible enough to accommodate runtime changes to improve design can be efficient, and even monetization.

All the tasks easily. Game controls should be easy to spot and understand. For Android touch devices, the placement of controls is also very important.

Technical design optimization

for the project

.

The technical design document also specifies the scope and scale of the game. Such specifications help run the game smoothly on a device, because the hardware platform is already covered within the technical design document.

This is a pre-development process. A few assumptions need to be taken care of in this document. These assumptions should be optimized enough to evolve when a real-time situation occurs.

development.

By optimizing these tasks, it is much easier to implement and execute:

- Program architecture
- System architecture
- System characteristics
- Defined dependencies
- Impacts
- Risk analysis
- Assumptions

with effort,
time.

Memory optimization

Memory has its physical limitation based on the hardware configuration, but games and applications cannot be made separately for each device.

targeted hardware platforms should be mentioned. Now, it is a very common scenario that an application is crashing. The developer is awarded with a memory overflow exception.

To avoid this scenario, there are two main things to be taken care of:

- Keep memory peak within the defined range
- Don't keep data loaded in memory unnecessarily

Android does not offer memory swapping. Android knows where to find the paged data and loads it accordingly.

Here are some tricks to optimize memory in Android gaming.

Don't create unnecessary objects during runtime

It leaves little:

```
//Let's have an integer array list and fill some data
List<int> intListFull = new ArrayList<int>();
//Fill data
for( int i = 0; i < 10; ++ i)
{
    intListFull.add(i);
}

// No we can have two different approach to print all
// values as debug log.
// Approach 1: not optimized code
for ( int i = 0; i < intListFull.size() ; ++ i)
{
    int temp = intListFull.get(i);
    Log.d("EXAMPLE CODE", "value at " + i + " is " + temp);
}
// List size will be calculated in each cycle, temp works
//as auto variable and create one memory footprint in each
//loop. Garbage collector will have to clear the memory.

// Approach 2: optimized code
int dataCount = intListFull.size();
int temp;
for ( int i = 0; i < dataCount ; ++ i)
```



```
{
    temp = intListFull.get(i);
    Log.d("EXAMPLE CODE", "value at " + i + " is " + temp);
}
// only two temporary variable introduced to reduce a foot
//print in each loop cycle.
```

Use primitive data types as far as possible

User-defined data types take more memory space than primitive data types. In

Android, if the developer uses the `Integer` class instead of `int`, the data size increases four times.

For Android compilers (32 bit), `int` consumes 4 bytes (32 bit), and `Integer` consumes 16 bytes (128 bit).

With full respect to modern age Android devices, limited use of this data type may cause no significant harm to memory. However, extensive use of non-primitive data types may cause a significant amount of memory block until the developer or garbage collector frees the memory.

So, the developer should avoid `enum` and use static final `int` or `byte` instead. `enum`, being a user-defined data type, takes more memory than a primitive data type.

Don't use unmanaged static objects

In older Android versions, it is a common issue that a static object does not get destroyed automatically. Developers used to manage static objects manually. This issue is no longer there in newer versions of Android. However, creating many static objects in games is not a good idea as the life span of static objects is equal to the game life. They directly block memory for a longer period.

crashing
the game.

Don't create unnecessary classes or interfaces

Each class or interface has some extra binding space in its instance. The modular programming approach demands maximum possible breakage in the coding interfaces.

This is considered to be a good programming practice.

However, this has a consequence on memory usage. More classes consume more memory space for the same amount of data.

Use the minimum possible abstraction

ing
ary and provide
only selective APIs. When it comes to game development, if the developer works on games only, then use of abstraction is not very necessary.

Abstraction results in more instructions, which directly leads to more processing time and more memory use. So, even if abstraction may be convenient sometimes, the developer should always think twice before using abstraction while developing games.

For example, a game may have a set of various enemies. In such a case, creating a
elps create
ely
ill depend on
en it will
always increase the set of instructions to be processed at runtime.

Keep a check on services

t they are
very costly in terms of both process and memory. A developer should never keep a
e service
life cycle is to use `IntentService`, which will finish once its work is done. For
other services, it is the developer's responsibility to make sure that `stopService` or
`stopSelf` are being called after the task is done.

This process proves to be very efficient for game development, as it actively supports dynamic communication between the user and developer.

Optimize bitmaps

Bitmaps are the heaviest assets for a game. In game development, most of the heap memory is used by bitmaps. So, optimizing bitmaps can significantly optimize the use of heap memory during runtime.

n by this
formula:

$$\text{BitmapSize} = \text{BitmapWidth} * \text{BitmapHeight} * \text{bytePerPixel}$$

For example, if a 480 x 800 size bitmap is being loaded in the ARGB_8888 format (4 bytes), the memory will be as follows:

BitmapSize = 480 x 800 x 4 = 1536000 bytes ~ 1.5mb

The format can be of the following types in Android:

- ARGB_8888 (4 bytes)
- RGB_565 (2 bytes)
- ARGB_4444 (2 bytes) (deprecated in API level 13)
- ALPHA_8 (1 byte)

It is

unnecessary heap usage.

Release unnecessary memory blocks

As we have discussed earlier for freeing memory, the same can be applied on any object. After the task is finished, the instance should be set to null so that the garbage collector can identify and free the allocated memory.

to free the
the member
should be a bad idea
selectively free
the memory of unused objects without deleting the class instance.

Use external tools such as zipalign and ProGuard

The ProGuard tool is efficient at shrinking, optimizing, and obfuscating the code by removing unused code and renaming classes, fields, and methods with a secured and encoded naming structure. ProGuard can make the code more compact, which directly impacts RAM usage.

aries,
which may be pre-compiled with ProGuard. In those cases, the developer must
conflict the
codebase from getting stolen.

further to
use less space and have a more compact size. Normally, most of the APK building
need to
use it manually for few cases.

Performance optimization

and

droid

gaming, we already know about the wide range of hardware configurations.

Maintaining the same performance across all devices is practically impossible. This is the reason developers choose target hardware and minimum hardware configuration to ensure that the game is performing well enough to be published. However, the expectation also varies from device to device.

the targeting

set of hardware. Thus, memory has its own optimizing space in the development process.

n can be

done by paying more attention to writing and structuring code:

- Using minimum objects possible per task
- Using minimum floating points
- Using fewer abstraction layers
- Using enhanced loops wherever possible
- Avoiding getters/setters of variables for internal use
- Using static final for constants
- Using minimum possible inner classes

Using minimum objects possible per task

Creating unnecessary objects increases processing overhead as they have to be task

multiple times is much faster. Here is an example:

```
public class Example
{
    public int a;
    public int b;

    public int getSum()
    {
        return (a + b);
    }
}

//Lets have a look on un-optimized code
// Here one object of Example class is instantiating per loop //cycle
```

```
// Same is freed and re-instantiated
public class ExecuterExample
{
    public ExecuterExample()
    {
        for ( int i = 0; i < 10; ++ i)
        {
            Example test = new Example();
            test.a = i;
            test.b = i + 1;
            Log.d("EXAMPLE", "Loop Sum: " + test.getSum());
        }
    }
}

// Optimized Code would look like this
// Here only one instance will be created for entire loop
public class ExecuterExample
{
    public ExecuterExample()
    {
        Example test = new Example();
        for ( int i = 0; i < 10; ++ i)
        {
            test.a = i;
            test.b = i + 1;
            Log.d("EXAMPLE", "Loop Sum: " + test.getSum());
        }
    }
}
```

Using minimum floating points

In machine-level language, there is nothing like an integer or float. It is always a bit er can be directly represented by a set of bits, but floating points requires extra processing overhead.

Until a point of time, there was no use of floating points in programming languages. Later, the conversion came, and floating point was introduced with extra processing requirements.

Using fewer abstraction layers

So, as we increase the abstraction layers, the process becomes slower.

Using enhanced loops wherever possible

In the case of array and list parsing, an enhanced `for` loop works way faster than the usual conventional `for` loop as it has no iterating variable system, and each array or list element can be accessed directly.

Here is an example of a non-enhanced loop:

```
int[] testArray = new int[] {0, 1, 2, 3, 5};
for (int i = 0; i < testArray.length; ++ i)
{
    Log.d("EXAMPLE", "value is " + testArray[i]);
}
```

Here is an example of an enhanced loop:

```
int[] testArray = new int[] {0, 1, 2, 3, 5};
for (int value : testArray)
{
    Log.d("EXAMPLE", "value is " + value);
}
```

Avoid getter/setters of variables for internal use

al element
ot follow the
ed widely in
Android game development.

In many cases, developers use getters and setters from inside the class object. This unnecessarily increases processing time, resulting in degraded performance. So, developers should use getters and setters as little as possible and make sure they are not being used internally.

Use static final for constants

bal
we're required
to parse the class object in order to access it.

ent accessibility

increases significantly when using static for constants. However, the developer needs to keep a check on memory usage as well.

Using minimum possible inner classes

Each inner class adds an extra layer to processing. Sometimes, it is good to have inner classes in order to structure the codebase in an efficient and readable way. However, it comes with the cost of processing overhead. So, the developer should use the fewest possible inner classes in order to optimize performance.

Relationship between performance and memory management

In Android game development, performance and memory optimization often conflict with each other. To maintain the visual quality of the game, better art assets are mandatory, which eventually increases memory overhead and performance lag.

Optimizing memory needs to do frequent memory operations, resulting in performance drop. To increase performance, objects have to be readily available for ls.

Balancing between them is the only way out to optimize the full game to run smoothly without exhausting memory.

Memory management in Android

Let's discuss the memory management system in Android. It has a direct effect on roid.

zed

states of the game. There are three main topics to discuss to understand the working principles:

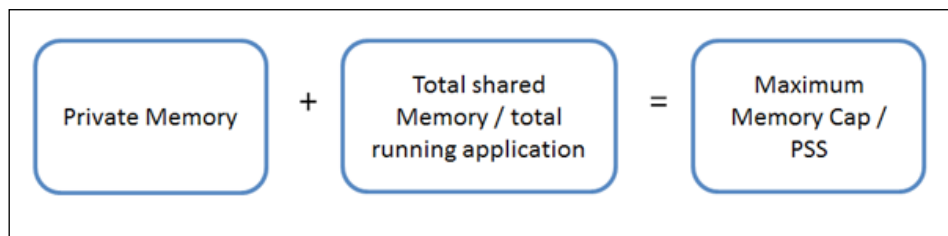
- Shared application memory
- Memory allocation and deallocation
- Application memory distribution

Shared application memory

the same

memory segment within running processes or services. For example, Android often shares the "code" memory within processes. Very often, external libraries and JVM's executable code memory can be safely shared across processes without creating a new process. If a process modifies the shared memory.

Android allocates dedicated memory for each application or process. This is called private memory. The same process may also use shared memory. Android then the total memory used by the application is the sum of private memory and shared memory. This cap is called **Proportionate Set Size (PSS)**:



the process

is kept

in the background

activities or services to carry out some task. The developer has to make sure that the application uses minimum possible memory at any point in time, especially when the application is in the background.

any shared

memory

to reduce

the chances of your application getting unexpectedly killed by the Android system.

Memory allocation and deallocation

The Android memory management system defines a virtual cap for each application, which is the total memory used by the application. If there is free memory, the application can allocate more memory. The memory used by the application varies during runtime and shared memory dependency.

Application memory cannot use more physical memory than PSS. So, after reaching it will receive

`OutOfMemoryError` thrown by the system. Android might kill other empty or background processes to accommodate memory for the running application in a few cases:

- If the application quits
- If the process becomes inactive and some other process requires the memory
- If the application crashes for any reason

Application memory distribution

It is a tasking

environment. The exact heap size limit varies between hardware configurations and the heap

capacity and tries to allocate more memory, it will receive `OutOfMemoryError`, and the application will be killed by Android.

The developer needs to check the amount of memory available on the device using the

operating system for this amount of memory by calling `getMemoryClass()`. This returns the application's heap.

Processing segments in Android

Applications or

games can run on an Android platform. However, for games, only one game is active at one point of time, but rest of the applications run in the background.

Let's have a look at how Android processes its applications.

Application priority

running

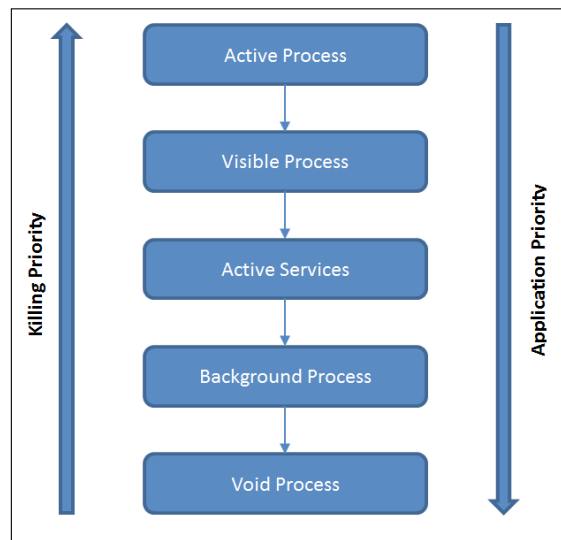
application of low priority depending on the requirement.

Each application uses some memory and processing bandwidth. There may be a situation where multiple applications are running together. If a new application wants to run, then Android allocates memory and process bandwidth for the new application. Android kills

one or more than one running application with low priority.

Android sets priority by the following status:

- Active process
- Visible process
- Active services
- Background process
- Void process



Active process

An active process is basically a process that communicates with the platform very often and is not killed by Android, when necessary.

An active process fulfils the following criteria:

- It runs in the foreground
- It is visible
- At least one Android activity is running
- It interacts actively with the user interface
- All event handlers are in the active state

Visible process

This process is basically an active process that is not in the foreground and does not interact with the user interface. It is the second highest priority for the Android platform.

The criteria for this process are as follows:

- It runs in the background
- It has visible activity
- It does not interact with the user interface
- UI event handlers are not active
- Process event handlers are active

Active services

visible

interface. Android will kill such services first and then the actual active process.

This service follows the following criteria:

- It has no visible interface
- It supports or works for respective active processes
- It runs in the background

Background process

Background processes are basically minimized or inactive processes. These processes are not visible on the screen. The process thread does not run for these processes, but they are killed by the processor. These processes can be resumed after interruption.

These are inactive/minimized processes. They remain in memory. The application stays in the paused state.

Void process

Void processes are also called empty processes. A void process is literally empty. It holds no application data or state in memory. This process has the highest priority in order to get killed by the operating system.

Application services

These services may run within and outside the parent process.

Let's clear two very common misconceptions about services:

- A service is not a separate process
- A service is not a thread

e processes.

background,

and they keep running even if the main application is in a suspended state.

arent

ed.

Service life cycle

Services are started by the parent application process, as follows:

```
Context.startService();
```

After being started, the service starts carrying out a single task in the background.

e file download

service will stop after a successful downloading task. Many game developers use such features in their games to improve the user experience.

These services can be bound with one or more processes for interactivity. The application can send request and get response from a bound service, which creates a time until the

last application component is bound with the service.

Resource processing

Android has its own resource process structure. It has some predefined resource types:

- Drawable resources
- Layout resources
- Color resources
- Menu resources
- Tween animation resources
- Other resources

Drawable resources

Android

provides the `res/drawable/` project path dedicated to all drawable resources. All bitmaps, various XML, and predetermined frame animations can be placed here.

These can be accessed through the `R.drawable` class.

Layout resources

All defined layouts fall in this category. Android provides the `res/layout/` project path dedicated to all layout files. Layout is useful to define the application UI.

These can be accessed through the `R.layout` class.

Color resources

on changing

the view of the applicable object. Android stores this in the `res/color/` folder in the hierarchy.

These can be accessed through the `R.color` class.

Menu resources

All menu contents can be defined here. Android provides the `res/menu/` project path dedicated to all **drawable** resources.

These can be accessed through the `R.menu` class.

Tween animation resources

All tween animation resources fall in this category. Android provides the `res/anim/` project path dedicated to all tween animation resources.

These can be accessed through the `R.anim` class.

Other resources

All other resources are places in the `res/values/` folder. Many developers define the string under this category with styles.

These can be accessed through the `R.values` class.

Different memory segments

s are
used depending on the behavior:

- Stack memory
- Heap memory
- Register memory

Stack memory

ed in the
use. So,
there is no manual memory management process associated with the stack memory.

However, extensive use of auto variables also may cause memory errors. This is the reason we have already discussed why minimizing unnecessary auto variable declarations is necessary.

Stack memory is also used to execute program instructions. Each instruction is broken down into a single operation and put into a stack by the interpreter. Then, a recursive procedure is used to execute all the instruction stacks and return the result.

Let's have a look at how stack memory works for objects and primitives:

```
public class ExampleClass
{
    public ExampleClass()
    {
        int bitMapCount = 0; // primitive type
        Bitmap testBmp = BitmapFactory.decodeFile("bitmap path");
        // Object loading
        bitMapCount = 1;
    }
}
```

In this example, `bitMapCount` is an `int` local variable and gets stored in the stack scope.

However, `testBmp` is a bitmap object, which will be allocated in the heap, but the
t of the
ector can

identify the heap memory allocated for `testBmp` as having zero reference and will free this memory segment.

Heap memory

Heap memory is the segment where all the instances of classes and arrays are stored. JVM allocates this memory while instantiating any object.

ent
the memory
the memory
tion.

t. Art is
the most significant asset among them. So, optimizing bitmaps has a direct impact on heap memory uses. Very often, the developer allocates memory for assets and does
ing the
entire runtime.

Here is an example:

```
// create a bitmap in a class constructor having global
// scope with public access
public class ExampleClass
{
    public Bitmap testBmp;
    public ExampleClass()
    {
        testBmp = BitmapFactory.decodeFile("bitmap path");
    }
}
```

he use, until
the ExampleClass instance is there in memory. The interpreter has no standing
instruction to free the memory segment, because testBmp still has the reference to
the memory allocated to the bitmap.

We can optimize this in the following way with a bit of modification:

```
public class ExampleClass
{
    public Bitmap testBmp;
    public ExampleClass()
    {
        testBmp = BitmapFactory.decodeFile("bitmap path");
    }
    // create a method to free memory allocated for the
    // bitmap after use
    public void unloadBitmap()
    {
```

```

        testBmp = null;
    }
}

```

In this case, by calling `unloadBitmap()` after the use of the bitmap will remove the reference of the loaded bitmap from `testBmp`. So, the garbage collector will find this memory location as zero-referenced memory and free it to be used for other allocations.

Register memory

egister

memory. Registers are directly associated with the processor, and the processor stores the most significant and frequently used data in this memory segment.

untime.

Importance of memory optimization

, if the

game does not run on the target platform, then it cannot be successful. We already know that Android has various sets of hardware configurations.

The main variations of hardware are specific to the processor and memory. In the case of processors, it depends on their speed and quality. In case of memory or RAM, it is only the volume.

Even today, RAM can vary from 512 MB to 4 GB in Android devices. Memory optimization should always have a minimum target of RAM as per design. So,

minimum available RAM.

Sometimes, the developer fits the peak usage within the target limit of memory.

-time

scenario most of the time. There is always an error margin. So, it is not always true ded with the

same memory. This is the place when memory optimization plays a major role. It helps a lot in creating the buffer range for the game to run in a real-time scenario.

when

it does not require the amount of RAM it demands. This clearly indicates that the he most

common problems in game development. Optimizing memory properly helps get rid of this problem.

the game to stay in the background. When an application goes into the background, Android reground es the minimum possible memory while running. So, it is possible to save the data of the less memory.

Many games use game services at the backend. If the application is not active, then ting system.

Optimizing overall performance

Earlier, we discussed performance optimization from only the programming point of view. Let's discuss other scopes of optimizing the performance of Android games.

The developer can optimize performance from the time of design to development through the following points:

- Choosing the base resolution
- Defining the portability range
- Program structure
- Managing the database
- Managing the network connection

Choosing the base resolution

From the point of view of game development on Android, choosing the base resolution is probably the most significant design decision. Base resolution defines that the developer chooses to work upon, the more storage and process time it takes. Base be stored with the n increases, ificant influence on processing.

ng bigger and better. So developers now choose a bigger resolution to support higher range devices.

Defining the portability range

This is also a design phase optimization. In this stage, the developer needs to decide the range of hardware platform to support. This includes various configurations. riations in terms of memory, processing speed, graphics quality, and so on.

If the range supports the range of portability of a similar device, then optimization becomes easier. However, this is not the case for most cases of game development. s:

- Low-performing devices
- Average-performing devices
- High-performing devices

fine the portability range.

Program structure

n for both performance and memory optimization. This includes all the parameters for programming optimization, which we have already discussed.

developer

A few

singleton classes help here to optimize performance significantly. Proper game state machine design also helps optimize performance.

Managing the database

There are many games that are mainly data driven. In such cases, a database needs to be managed properly.

abase at

take time

er sends a

query to the database. Then, the database fetches the data, binds it accordingly, and a in order to

rformance

overhead. Using a faster database also helps the game to perform well.

Managing the network connection

Modern day gaming has enhanced to multiplayer and server-controlled mechanisms, es, network types of multiplayer architecture currently being followed:

- Turn-based multiplayer
- Real-time multiplayer

eal-time
ultiplayer.

ent on the
imized in order
to achieve the following goals:

- Less lag time
- Less layer processing
- Less number of pings to the server

Increasing the frame rate

e rate. A high
r has to
smoothness and
effect.

me
or mid-scaled 3D game is considered high performance. On the other hand,
massive 3D games might consider an average FPS of 30-35 as good performance.

ects to
improve the user experience. This has a direct impact on monetization.

Importance of performance optimization

As we have just discussed, performance optimization directly influences the frame
formance
optimization has other importance too:

- Games might crash or go in a not-responding state due to a non-optimized program
- Performance optimization has a direct impact on memory as well
- Performance optimization can enlarge the range of supported hardware platforms

Common optimization mistakes

pick up with
optimization
commits the
following mistakes knowingly or unknowingly:

- Programming mistakes
- Design mistakes
- Wrong data structure
- Using game services incorrectly

Programming mistakes

Programming is a manual process, and to err is human. So, it is obvious that there are a few ways in which a programmer can minimize mistakes to have an optimized game code base. Let's discuss the major mistakes a programmer commits while developing a game in Android.

Programmers often create many temporary variables and forget to keep track of them. Often, these variables occupy unnecessary memory and increase processing calls.

Sorting is widely used in game development for many purposes. There are several sorting algorithms. Most of the time, the developer chooses convenient techniques rather than efficient ones. For large arrays or lists, this may cause a serious lag in process flow.

Using too many static instances for accessibility ease is another bad practice. Using many static instances, as it blocks a lot of memory space during its lifetime. Many programmers even forget to manually free this memory.

ower.
ammings,
it only helps in limited cases.

are
several ways to work with loops. A programmer should first determine what goes best with the algorithm.

may take
time to build up the perfect logic for certain tasks. Many game programmers do not great scope of
optimization unexplored.

Design mistakes

Designers often make mistakes when defining the hardware range and the game n.

Another mistake is to target the wrong target resolution. The target resolution has a direct effect on the art asset size. Targeting the wrong resolution leads to unnecessary scaling, causing extra processing overhead.

Wrong game data structure

ts
data. Lists are
much slower than arrays. Lists should only be used when it is absolutely necessary.

It is the developer's responsibility to figure out the perfect data structure for data-driven games. Proper technical design should include a data structure model and its use.

Using game services incorrectly

rvices are
used for download/upload of data, for push notifications, for deep linking in games, or for server connectivity. However, services come at a huge cost of processing and memory consumption. Running services causes significant amount of power consumption as well.

So, using services should be mandatory only when there is no other way around.

Best optimization practices

Some defined and logical optimization techniques are available. We will discuss the major scopes and fields that are related to Android game development:

- Game design constraints
- Game development optimization
- Game data structure model
- Using game assets
- Handling cache data

Design constraints

It is always a best practice to define the target hardware platforms and acknowledge its according to it.

signing
along

with other constraints. We have already discussed design optimization. All those segments should be evaluated before going into development.

Targeting screen size and resolution has to be fixed when designing the game along with creating layouts, which will fit in multiple resolutions. This is because Android has many screen sizes as discussed earlier.

Selecting the minimum Android version and the target Android version gives the sorted API levels and platform features are already defined.

Development optimization

Here are some tips to carry out the development process successfully with optimization:

- Using as many as possible folder structures provided by Android for project scalability.
- Using resource formats according to the dpi list provided by Android.
- The developer should avoid scaling images. This effectively reduces memory and processing overhead.
- Using sprites for multiple purposes is also a good practice to create animations.

- The tiling technique is very useful in terms of reducing memory consumption.
- Overriding the `onDraw()` method is always a good practice to flush the requirement.
- Use XML-based layout wherever possible; however, games have very limited scope for this Android feature.

Data structure model

Since the
uses data for
various purposes such as sorting, searching, storing, and so on.

There are many data structure models available for various operations. Each operation has its own advantages and disadvantages. The developer must choose the most efficient one depending on the requirement.

linked
list. Effectively, linked lists are more flexible and dynamic in nature than arrays. However, this feature comes at a cost of slow processing and higher memory consumption.

e, if
cricket team data needs to be stored, then an array is sufficient, because there will always be 11 players on each side, and that cannot be modified during gameplay. It will make the process much faster and more efficient than using a linked list in this particular case.

number of
bullets the user may fire during gameplay. So, a queue data structure will be most efficient in order to process all the fired bullets.

Similarly, stacks and tree structures can be chosen whenever they fit the purpose. The same approach may be taken for sort and search algorithms.

Asset-using techniques

We have already categorized assets for games. Let's discuss them from the perspective of optimization techniques and best practices.

Art assets

Art assets are enough to start gameplay.

As we have discussed already, better art assets cost memory and performance. Less for art assets, which eventually results in poor visual quality.

Art should never compromise from the perspective of visual quality. Often, artists develop assets that do not reflect perfectly in games because of inappropriate optimization.

We have already discussed how art assets should be made. Now, let's assume that exported in a 3D format to a typical 8-bit format without affecting the visual quality.

Removal of the transparency information in order to have optimized art assets.

Audio assets

Audio assets are standalone assets too. Audio has become a very important asset for extended user experience. Audio configuration can vary with a wide range of frequency, bit depth, and compression techniques. Each variation in configuration has a different level of processing and memory consumption.

Processing of audio assets into different formats of audio for SFX and music files.

One thing that developers generally ignore is audio information data. Few Android developers are aware of the cap for Android game sounds. So, every sound should be made within the proximity.

Sound designers need to keep up the quality within the limit. In this way, audio assets can be optimized at the time of development.

Other assets

Besides art and audio, there may be other data assets used in games. The data format can be anything, such as binary, text, XML, JSON, or custom. Custom formats are basically the same as binary format, with some encryption.

It is a common practice in game development to use data sets separately. A separate data set helps structure the project and give flexibility to use the same code for a different output. Often, the developer updates data source to update the complete e in the longer run in order to maintain the game and do easy updates.

zed enough to get processed quickly and not consume too much memory. However, reading and writing an external file takes time. Normally, binary files are the fastest to be processed and smallest in size. However, after reading the binary data, it has to be parsed to be used in games, which eventually increases processing.

The most commonly used data formats are XML and JSON. The Android library has can have readily available data without making extra processing effort. However, the data can be manipulated during gameplay, depending on the game's requirements.

Handling cache data

A cache is a memory segment that is similar to RAM from a functionality point of view, but acts faster than conventional RAM. The processor can access this segment much faster. So, logically, a cache should only store data that is being used frequently.

lication
ailable for
e of 2% of the
total free memory.

ly make
so that
the executer automatically uses cache memory for them.

Summary

ent,
amming.
nical

programming as it has the most common algorithms to implement. However, in the orithms.

In many cases, an artificial intelligence algorithm is also made separately. So, there is a very high probability that the programmer has to find out an efficient way to optimize freshly written algorithms.

We have discussed all the possible scopes of optimization in Android game development. Technical optimization is mandatory as it has fixed guidelines to follow. However, logical development will depend on the game algorithm and its requirements. So, it is an extra effort for the game developer to optimize Android games.

Sometimes, developers over-optimize games. This is not recommended.

time of technical design, optimization cases should be declared.

Most large-scale development processes have a separately defined task set for optimization. Some developers choose to develop a dynamic optimization process. different

scales. Both the processes are effective, but the first one is logically more sensible, because defining a separate task will always give an idea about the tentative time elopment process in a better way.

sign,

We will have a deeper look at testing in the next chapter of this book.

9

Testing Code and Debugging

ustry.
ally not
and
pport the
most platforms with the maximum possible efficiency.

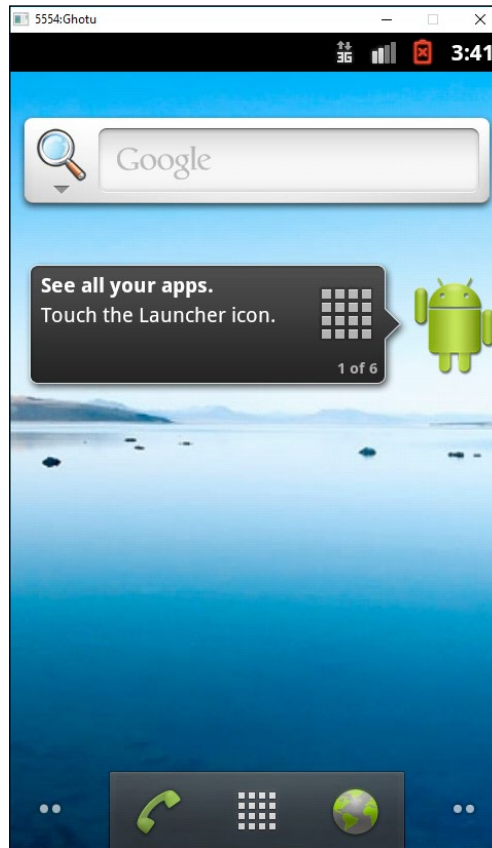
We will discuss the scope of various debugging aspects in Android game development through the following topics:

- Android AVDs
- Android DDMS
- Android device debugging
- Monitoring the memory footprint
- Strategic placement of different debug statements
- Exception handling in Android games
- Debugging for Android while working with cross-platform engines
- Best testing practices

Android AVDs

AVDs are the most significant and important part of debugging Android games.
few predefined
oid
emulator provides an interface of a real-time-like device.

AVDs have a few features that virtually provide the device RAM, Android version, screen size, display dpi, keyboard, and different visual skins. Older AVDs mostly looked the same.



In the current version of Android Studio, most of the Android device categories are provided. Developers can create AVDs as per the target development platform.

The categories are as follows:

- Android mobile phones
- Android tablets
- Android TVs
- Android wearables

in
VD

manager. This tool can also help the developer to create a custom AVD.

Let's have a look at the attribute factors for each different AVD:

- Name
- Resolution
- Display size
- Android version API level
- Android target version
- CPU architecture
- RAM amount
- Hardware input options
- Other options
- Extended AVD settings and creation

Name of the AVD

It can be changed later. Predefined AVD names can also be changed at the time of creation.

AVD resolution

There are some predefined platform.

Many of games are tested on various resolutions to check the compatibility.

Normally, multiple resolutions would create any issues if the aspect ratio is the same. However, in the case of Android, we can find multiple aspect ratios for different devices. The resolution factor of AVD helps fit the game and check its compatibility for multiple aspect ratios as well.

AVD display size

solution AVD
gh dpi value,
which means a higher display quality.

, it is not
em as the
development system has its own display limit.

Android version API level

sage
to a certain version. The API version can be deprecated in future versions of
Android or even discontinued. To check this factor, the developer can set an API
version for AVD.

Android target version

This is the Android version that will be used to run the AVD. This can verify the
manifest target Android version and minimum version range.

CPU architecture

Android devices mainly use three types of CPU architecture: armeabi, armeabi-v7,
and x86. This does not have a direct impact on games. However, the processing
speed and quality varies with CPU architecture.

than an
to be tested
on a real device.

RAM amount

RAM amount specifies the total amount of memory that the AVD has, which can be
used to check the memory consumption of the game at various levels.

It is best to predict the memory overflow issue for various devices. By running
The default
value is set to 66 MB. The developer can set any value according to the requirement.

External storage can be also defined as an SD card for a virtual device.

Hardware input options

d within a wide range of hardwires. The most common variations are as follows:

- Touch screen
- Touch pad
- Key pad
- Custom controller
- Hardware buttons

An AVD creates a virtual system for all of these input systems.

Other options

n. If the e a camera, both front and back.

Additionally, virtual accelerometers, sensors, and so on can be associated with an AVD.

Extended AVD settings

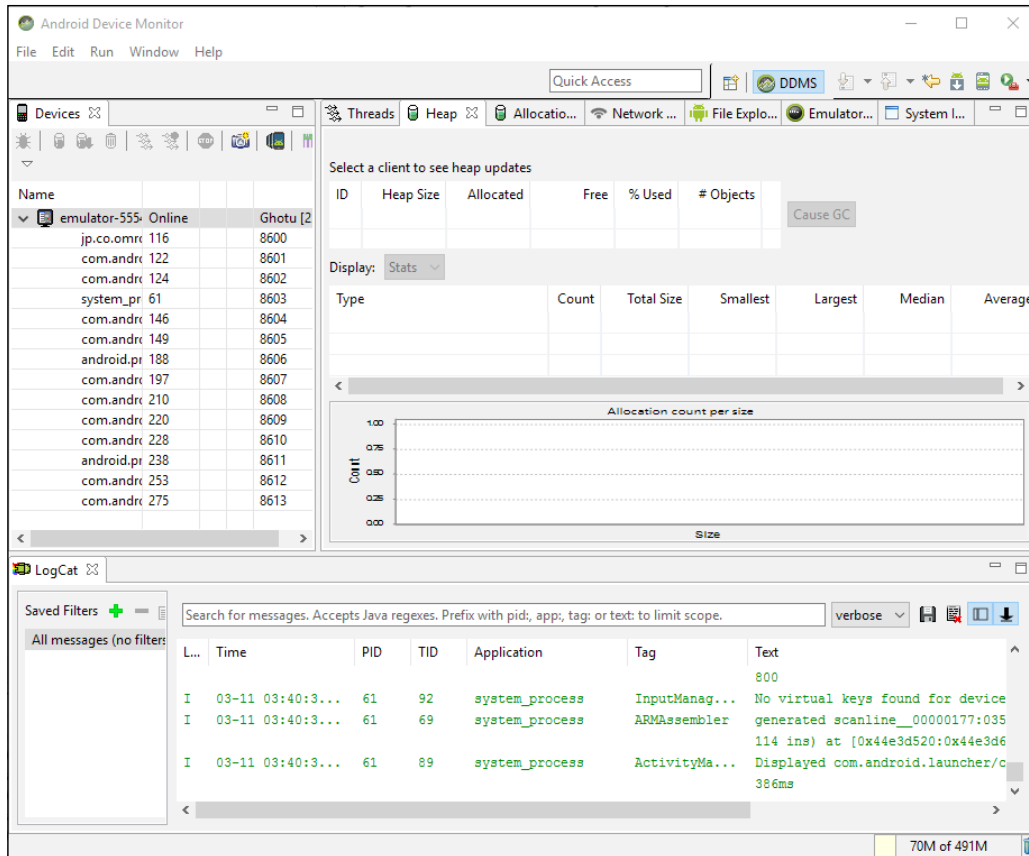
can design a virtual device with a custom look and feel, and with complete custom hardware configuration.

Android DDMS

Is such as memory consumption, process calls, and so on.

The main functions of Android DDMS are port providing, screen capture on a device, thread details, heap details, and Logcat processing. This service can be used for spoofing calls and messaging.

Android DDMS is widely used for device debugging. Particularly in the game em. This
ack runtime
exceptions:



Android DDMS can be used to carry out the following activities.

Connecting an Android device filesystem

DDMS can connect to a device filesystem and provide a file browser-based operation to copy, modify, or delete any file on the device through a PC. This method or feature, however, is not very important for Android game development.

Profiling methods

Another interesting DDMS feature is profiling or tracing matrices of certain methods. It gives information on the following topics:

- Execution time
- Number of operations
- Number of cells
- Memory use during execution

Extending this feature, the developer can even gain control over profiling the data of a method by calling `startMethodTracing()` and `stopMethodTracing()`.

The developer needs to keep an eye on two things:

- Up to Android 2.1, it is mandatory to have an SD card installed on the device with the application's permission to read/write on it
- From Android 2.2 onward, the device can stream profiling data directly to the development PC

Thread information monitoring

DDMS provides details on each thread running for each process on a selected device. However, games mostly run on a single thread. As devices are getting better each operations such as rendering, processing, file I/O, and networking.

Heap information monitoring

Useful for game developers to track the game process heap during execution.

Tracking memory allocation

time objects.

This gives every detail on each specific object of each class. This means the developer can find out which class is taking exactly how much memory. This helps achieve memory optimization in a much more efficient way.

Monitoring and managing network traffic

From Android 4.0 onwards, DDMS features a **Detailed Network Usage** tab to per can e network development. It can distinguish between different traffic types by applying a "tag" to network sockets before use.

Tracking log information using Logcat

Logs are the most useful debugging technique for tracking almost anything. It is a objects during runtime. It is very useful for logic development for games.

t games. So, there must be a good amount of code that is written for the first time. Predefined test cases are not available. This deficiency can be overcome using Logcat from DDMS.

Logcat provides log information in the following types:

- Verbose
- Debug
- Error
- Warning
- Information

Emulating device operations

As we discussed Android virtual devices earlier, DDMS can work upon AVDs as well. So, it becomes much easier to emulate a real-time scenario to debug the game being developed.

The most commonly used emulations are as follows:

- Emulating an incoming phone call
- Emulating an incoming message
- Emulating network state change during runtime

These three are the most common scenarios at runtime. So, these situations can be difficult for Android devices since the beginning. As a matter of fact, this can be a nightmare for a programmer if interrupts are not handled properly.

after
an interrupt. Many times, some unnecessary services or processes can be interrupts,
and they may change the game state during the interruption period. Emulating
up the
debugging or interrupt handling procedure.

Android device testing and debugging

Android device debugging is the most important part for any Android game development process. Let's divide this topic into two sections:

- Device testing
- Device debugging

Device testing

mber
of different devices. These different devices include different displays, different resolutions, different Android operating system versions, different processors, and different memory capacities. Due to these reasons, Android device testing is important and has to be carried out with great effort and planning.

Normally, in a game development cycle, first-point testing is carried out by the developer. This process makes sure that the game is running on devices.

s from various
aspects. This is the main part of device testing.

to game
development stages:

- Prototype test
- Full or complete test
- Regression test
- Release test or run test

In other words, a similar kind of distribution in each category is termed as follows:

- Pre-alpha test
- Alpha test
- Beta test
- Release candidate test

testing.
t's
describe these stages in brief.

Prototype testing

The developer and designer together develop a playable stage of the basic game idea during the phase of prototype testing.

Ideally, core gameplay is tested in this phase to analyze the feasibility, potential, and scope for the game concept.

Prototype testing is probably the most important part of the game development process. This phase determines the future of the game concept and also helps in developing a meta game and monetization model for the concept.

Full or complete testing

Usually, whenever the first few builds are submitted to testing in each phase, full testing is conducted. This reveals each and every possible issue with the game, and design faults.

lly implies
the possible completion time and effort for that game build.

Regression testing

producers
ct issues for
the testing
team for regression test.

In regression testing, a tester usually picks the issue and specifically checks whether it is actually solved or not. If the issue occurs in a fixed build, then the testers reopen the reported issues are addressed.

Release testing or run testing

his phase, the
just to check
this phase is
often called a "run test".

As many physical devices as possible are used for this segment of testing for a compatibility check. The final device support list is created after this testing phase. It is almost impossible to arrange all the available devices and perform a run test on them. So, the developer groups the devices according to their configuration and performance. Devices that behave in a similar manner are put in the same category, and only one or two devices are actually arranged for run testing for the whole group.

Device debugging

Now, we will commonly, it is done by both developers and testers.

In the game industry, device debugging is mainly used to find out runtime crashes, freezes, memory issues, networking issues, and performance issues. Through device debugging, the developer gathers the following information:

- Runtime maximum heap consumption
- Average FPS on various devices or multiple set of devices
- Unnecessary loaded objects
- Hardware button behavior
- Network request and response

Use of breakpoints

he game
gh DDMS.
uce
uation.

The developer can debug the logic line by line after a breakpoint so that the root cause of the behavior is found and fixed.

Monitoring the memory footprint

From
rint is
very important:

- Checking log messages
- Checking heap updates

- Tracking memory allocation
- Checking overall memory usage
- Tracking memory leaks

Checking log messages

flow and
runtime object tracking.

Dalvik message log

ollection
happens, the garbage collector can print the following information through Dalvik log messaging:

- **Garbage collection reason:** This info reveals the reason for triggering garbage collection. The reasons can be `GC_CONCURRENT`, `GC_FOR_MALLOC`, `GC_HPROF_DUMP_HEAP`, `GC_EXPLICIT`, or `GC_EXTERNAL_ALLOC`.
- **Amount of memory freed:** This section states the amount of memory freed by the garbage collector in KB.
- **Current heap memory status:** This shows the percentage of heap memory used and live objects memory/total heap.
- **External memory status:** There may be some operations that allocate memory externally. This section shows the allocated memory/garbage collection limit.
- **Garbage collector pause time:** Pause time is triggered twice, at the beginning in the case of a large heap.

ART message log

The ART message log is also capable of showing or tracking memory footprints. However, it is not triggered unless explicitly requested.

r takes more
case of ART,
the following information can be shown as logs:

- **Garbage collection reason:** In ART log messages, the developer can have Concurrent, Alloc, Explicit, NativeAlloc, CollectorTransition, HomogeneousSpaceCompact, DisableMovingGc, or HeapTrim as the reason for collection.
- **Name of garbage collector:** ART has few different garbage collectors that can be involved in a collection process. The name can be known by the field of the collection log. ART has these collectors: **Concurrent Mark Sweep (CMS)**, **Concurrent Partial Mark Sweep (CPMS)**, **Concurrent Sticky Mark Sweep (CSMS)**, and MarkswEEP plus Semispace.
- **Count of objects freed:** This shows the total number of objects freed from memory by the garbage collector.
- **Amount of memory freed:** This shows the total amount of memory freed by the garbage collector.
- **Count of large objects freed:** This shows the number of objects freed from the large object scope. These objects are freed by the collector.
- **Memory amount freed from large objects:** This shows the amount of memory freed from the large object scope. This memory is freed by the collector.
- **Current heap memory status:** This is the same as the one for Dalvik logs – live objects count/total heap memory.
- **GC pause time:** In the ART pause time section, this is directly proportional to the collector. Unlike Dalvik, the ART CMS garbage collector has only one pause time during the end of the collection process.

Checking heap updates

ure of the

There are

plenty of device memory monitors available in the market. DDMS device monitor
e game's
runtime.

The Android SDK comes with an inbuilt device monitor at `<sdk>/tools/monitor`.

The memory monitor in Android Studio is useful for Android Studio users. Monitors can interact with the Android application to watch heap update with each garbage ory usage for each segment of an application.

Thus, it becomes easier to optimize it further.

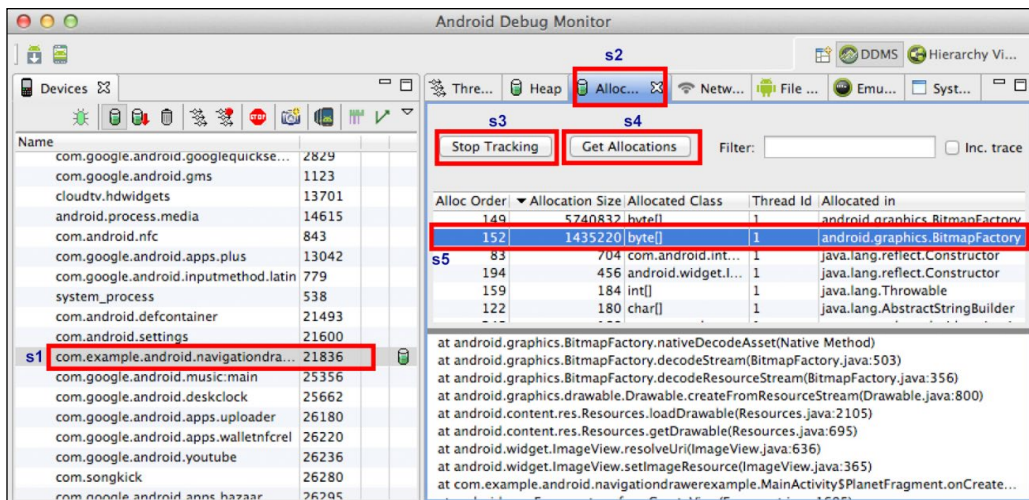
Tracking memory allocation

This is helpful for memory optimization. Memory allocation can be monitored through an **allocation tracker**.

ptimization.
ny useless
d remove
them for greater memory optimization.

id SDK
and Allocation Tracker in Android Studio.

However, it's not necessary to remove all allocations from performance-critical code paths; yet, the allocation tracker can help developers identify important issues in code. For instance, some apps might create a new `Paint` object on every draw. Moving this object into a global member is a simple fix that helps improve performance:



Let's have a quick look at the allocation information obtained:

- **s1:** This is the object package currently being tracked
- **s2:** This shows the **Allocation** tab is selected
- **s3:** This is used to start/stop tracking of the object
- **s4:** This updates the package allocation
- **s5:** This shows the allocation details

In game development, the number of objects in memory is immense, so it is very difficult monitoring tool helps find out the hidden spots that could have been easily ignored during the optimization process.

Checking overall memory usage

Overall memory usage of an Android game is distributed in different segments in RAM. This creates a general idea about application performance and memory security.

Basically, there are two types of allocation.

Private RAM

This is the dedicated memory portion used by the game during runtime. The Android operating system allocates this memory to the application. Private RAM is distributed in two segments:

- Clean RAM
- Dirty RAM

specific applications (in our case, it is an Android game).

Proportional set size (PSS)

This segment of RAM is used by multiple processes. It is basically shared memory. Due to its PSS value only in proportion to the amount of sharing.

Tracking memory leaks

solute

necessary to track memory leakage and resolve it. When a process allocates memory within the process.

age.

r can

always monitor memory consumption at any given point of time. A game runs different

he allocation/

deallocation of memory. Now, the developer can check the size of each object and hunt down the leakage. Another benefit of this process is finding unnecessary objects in memory alongside memory leakage.

Strategic placement of different debug statements

Anything and everything can be tracked and traced through debug statements. cost on

performance, which has a direct effect on runtime FPS. This is why a strategy on the placement of debug statements is absolutely necessary.

Let's have a look at the strategies related to following categories:

- Memory allocation
- Tracking the object state
- Checking the program flow
- Tracking object values

Memory allocation

er

ogramming

e collector

ay, a lag in

performance is observed.

Now, as a strategic placement to trace such mistakes, two debug messages should be placed at the constructor and destructor.

ful
nsumes.

Tracking the object state at runtime

An object can be initialized at any time during gameplay. Now, any external
. So, the object
y.

A successful debug statement and a failed debug statement (with reason) helps the developer rectify the issue.

o, the debug
statement identifies the spot. The developer can solve the issue with the help of
debug statements both for objects and program flow.

Checking the program flow

rogram
flow. A modular program can be tested with this system. Then, the module set can
be tested with one debug statement in each module start.

Any wrong or unnecessary calls can be removed or rectified through this process.
Proper program flow ensures a certain frame rate during runtime. So, this approach
can be used to optimize performance.

Tracking object values

t be correct.

So, putting a debug statement to check the loaded/initialized content is necessary
to avoid future conflicts.

tements
ule can

be designed using an object-tracking method, resulting in a better programming
structure.

Exception handling in Android games

Exception handling may not be a part of debugging, but it helps reduce the number of exceptions and unnecessary application crashes.

Exception handling in Android is the same as Java exception handling.

Syntax

Standard Java syntax for exception handling is as follows:

```
try
{
    // Handled code here
}
catch (Exception e)
{
    // Put debug statement with exception reason
}
finally
{
    // Default instruction if any
}
```

The suspicious code should be put inside a `try` block, and the exception should be handled in a `catch` block. If the module requires some default task to execute, then put it in the `finally` block. The `catch` and `finally` blocks might not be defined if there is no exception in each `try` block failure, which is a good programming practice. This process requires you to analyze the module to find out any vulnerable chunk of code. In some cases, you may have multiple default tasks.

This is the initial program design:

```
try
{
    // Task 1 which might throw exception
}
catch ( Exception e)
{
    // Handles exception
}
```

```
}  
finally  
{  
    // Task 2 which might throw exception  
}
```

The program should be written in this way:

```
void func1()  
{  
    try  
    {  
        funcTask();  
    }  
    catch ( Exception e)  
    {  
        // Handles exception  
    }  
}  
  
void funcTask()  
{  
    try  
    {  
        // Task 1  
    }  
    finally  
    {  
        // Task 2  
    }  
}
```

The developer should remember the following points:

- A try block can be used only with a catch block
- A try block can be used only with a finally block
- A try block can be used with both catch and finally blocks in sequence
- A try block cannot be used alone anywhere
- Nested try...catch is possible but not recommended as a good programming practice

Scope

There are plenty of predefined exception scopes depending on the exception type and cause. However, the major exceptions handled in a game development process are as follows:

- Null pointer exceptions
- Index out of bound exceptions
- Arithmetic exceptions
- Input/output exceptions
- Network exceptions
- Custom exceptions

Null pointer exceptions

opment.

`NullPointerException` is thrown when any null object is referred to in the code.

The developer should track the initialization and use of the object to rectify this issue.

Here is an example:

```
class A
{
    public int num;
    public A()
    {
        num = 10;
    }
}
// some method in other class which is called during runtime.
void testFunc()
{
    A objA = null;
    Log.d("TAG", "num = " + objA.num);
}
```

This will throw an exception as `objA` has been initialized with `null`. Hence, it does not exist. A modern smart compiler can detect this obvious exception during compile time, but the code might be like this, where we defined another class containing the `testFunc()` method:

```
class RootClass
{
    public A objA;
    public RootClass()
    {
        objA = null;
        testFunc();
    }

    void testFunc()
    {
        Log.d("TAG", "num = " + objA.num);
    }
}
```

In this case, most of the smart compilers cannot detect the upcoming exception. To handle this, the developer should add few more lines of code to the `testFunc()` method:

```
void testFunc()
{
    try
    {
        Log.d("TAG", "num = " + objA.num);
    }
    catch (NullPointerException e)
    {
        Log.d("TAG", "Exception:: " + e);
    }
}
```

Index out of bound exceptions

This exception is thrown when accessing an indexed address, which is supposed to be a part of contiguous memory allocation, but is not. The most common one is `ArrayIndexOutOfBoundsException` in the case of game development.

For example, if an array contains five fields and the program tries to access more than five fields, this exception will be thrown. Let's consider this piece of code:

```
int[] arrayNum = new int[5];
for ( int i = 0; i < 5; ++ i)
    arrayNum[i] = i;

Log.d("TAG", "arrayNum[5] is " + arrayNum[5]);
```

Here, the exception will occur in the log statement, as `arrayNum[5]` means the sixth element in the array, which does not exist.

Arithmetic exceptions

A mathematical expression can signify an undefined value, but in the programming aspect, "undefined" cannot be defined. Hence, `ArithmeticException` is thrown.

For example, if an interpreter tries to divide any value by zero, then the result becomes undefined, which is thrown as an exception. The same result can be seen when calculating the value of $\tan 90^\circ$.

A simple case might look like this:

```
void divideFunct(int num, int deno)
{
    try
    {
        Log.d("TAG", "Division Result = " + (num / deno));
    }
    catch (ArithmeticException ae)
    {
        Log.d("TAG", "number cannot divided by zero");
    }
}
```

Input/output exceptions

ware.
a read/
eed data to
lly stored in
a separate binary, text, XML, or JSON file.

Being separate files located at a particular path, these files can go missing, especially when those data files are downloaded from some other location, because there may be a connection interruption and the file may not get saved. In this case, when the game software tries to load such files, then `IOException` is thrown.

Let's look at a quick example:

```
try
{
    File dir = Environment.getExternalStorageDirectory();
    File objFile = new File(dir, "tmpPath/myfile.txt");
}
catch (IOException e)
{
    Log.d("TAG", "Error reading file :: " + e);
}
```

Network exceptions

Network exceptions may change at any point in time. Often, game developers ignore network errors, which causes crashing, freezing, or some malfunctioning in the running of the game.

Commonly handled exceptions are `HttpRetryException`, `UnresolvedAddressException`, and `NetworkErrorException`. If any HTTP request cannot be retired automatically, then `HttpRetryException` is thrown. If an application wants to connect to a certain address and the address is not found, then `UnresolvedAddressException` is thrown. `NetworkErrorException` is used to handle using the wrong protocol, and so on.

Custom exceptions

This is typically used for two purposes:

- Gameplay exception handling
- Game support tool exception handling

is a small scope for this exception in game development. This is not practiced by most Android developers.

ss.
quired.

Debugging for Android while working with cross-platform engines

Most
r this kind
of development.

Most engines come with a built-in profiler and provide some features to debug the game. However, the profiler feature is completely dependent on the manufacturer of the specific game engine.

eate a
wrapper to automatically switch from one platform configuration to another and display profiler details within a common user interface.

However, these cross-platform debug tools cost some extra processing and
l
with an error margin.

Best testing practices

or
d durability
after an application is published. The most common approach for Android game testing is manual testing.

However, this process is definitely not the best. As an Android developer, a unit test is always a best practice to save time and get accurate test results.

Tools and APIs

There are several tools and Android APIs that can be used to carry out the testing procedure. Some of them are inbuilt, such as Android Test Support Library, Dumpsys, Monkeyrunner, and so on.

d run
through Android Debug Bridge.

uch as click,
with the
following command:

```
adb shell monkey -p <Game Package Name> <Event Count>
```

Dumpsys provides status of the system during the runtime of an Android application. This can be triggered through the following command:

```
adb shell dumpsys <option>
```

, network
status, RAM uses, and more.

Testing techniques

utomated
procedure
in brief. Let's have a look at automated testing.

memory
ated. A
e test result,
and it is saved at a given location of the development system.

e of a game.
he application
program, such as elements, classes, and methods. Unit tests are further categorized
into two stages:

- Local test
- Instrumented test

Local test

This type of unit test works on a local machine and runs on the JVM. This saves a lot
amework or
limited dependency that can be satisfied with dummy objects.

Instrumented test

Instrumented tests have full dependency on the Android framework and must run on an Android emulator or on an Android device. This testing technique is used to and debug
e used
easily with dummy objects. The developer needs to define the testing object data before it can run the test in an Android environment.

Summary

Any development process is incomplete without quality and performance assurance. Testing is the phase of development where the game needs to be verified technically and logically to see whether it can perform in the real market.

The phases of testing, debugging, and profiling the game ensures the best possible quality of the game for the targeted Android platform range. Often, an Android game works on few Android devices but not on all targeted devices. The developer can identify and resolve the issues for some specific devices through a detailed testing procedure.

10

Scope for Android in VR Games

rds. So, a
trays an
experience about an imaginary or real environment with a digital computing system.

ents do
, gaming has
a new scope for exploration with the help of **virtual reality (VR)**.

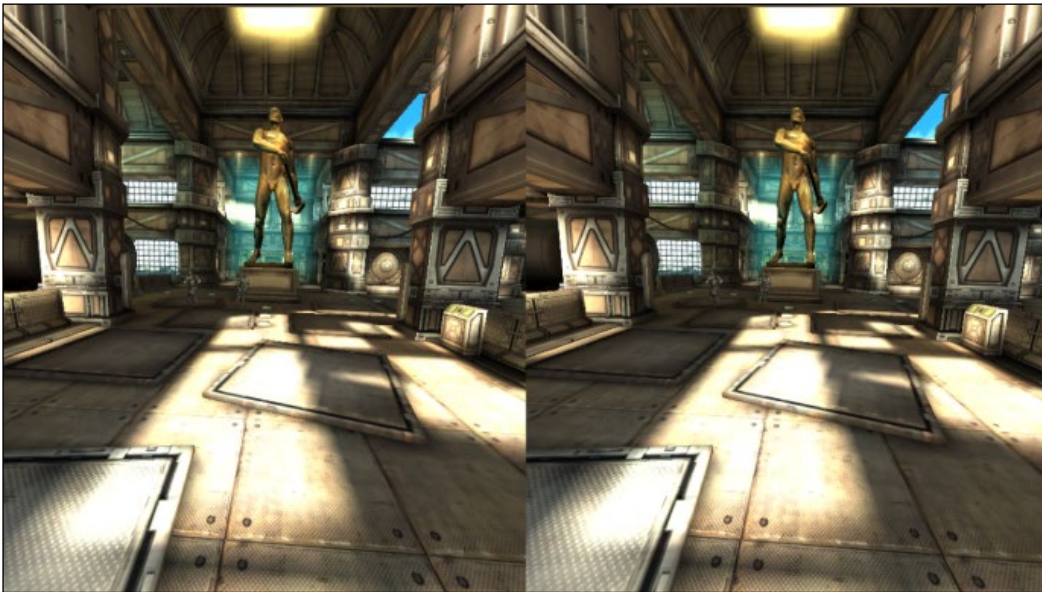
Let's explore the concept and scope for VR in Android games through the following topics:

- Understanding VR
- VR in games
- Future of Android in VR
- Game development for VR devices
- Introduction to the Cardboard SDK
- Basic guide to developing games with the Cardboard SDK
- VR game development through Google VR
- Android VR development best practices
- Challenges with the Android VR game market
- Expanded VR gaming concepts and development

Understanding VR

by
, but can
ry that

VR always replicates a real environment. It has the ability to replicate an imaginary world or environment, which can be displayed on a computer screen or a VR headset (head-mounted display) device (image source: <https://lh3.ggpht.com/uv8mx61-jsrbcu-EPNw1wIi4BCXg7338alepVlr7xKbKJf7eZ9EXT2U3roA8SWx1RC8=h900-rw>):



Actual screen display of the Shadowgun VR game

Evolution of VR

t the VR
concept was introduced around the second quarter of the twentieth century. In
1935, Stanley G. Weinbaum wrote a short science fiction story *Pygmalion's Spectacles*
graphic
recording of fictional experience with touch and smell. Technically, it defined virtual
reality.

ll, touch,
tudent Bob
Sproull created the world's first VR head-mounted display device.

Modern VR systems

Modern day VR devices evolved after 1990. They were lighter, with better display and consoles. This device was equipped with an LCD display, stereo headphones, and inertial sensors.

technical

ys, VR

devices are commercially available in the market. VR is now integrated in the latest mobile devices with input controller systems.

for

several purposes. Many more uses of VR technology are being introduced and improvised with each passing day.

Use of VR

VR is spreading each day in many sectors of the modern world. Let's have a quick look at the fields of VR usability:

- Video games
- Education and learning
- Architectural design
- Fine arts
- Urban design
- Motion picture
- Medical therapy

Video games

Video games are technically composed of display, sound, and various types of interaction systems. VR devices have proved to have all the necessary components for running a game application. Gamification of VR started in the late twentieth century. Since then, VR has been used in the gaming industry.

nvironment to

the game

world to interact with the elements.

chapter.

Education and learning

t on the learning process. Many times, it is not possible to provide a practical lesson on each educational

.

d to train pilots to fly fighter jets in developed countries. It reduces the chances of accidents, and trainee candidates can experience the real-time feel of flying a jet.

It is widely used in training medical students for various field treatments.

Architectural design

a proposed design without implementing it in reality. Many architectural firms use VR to demonstrate a design.

I copy of the architectural design.

Fine arts

This is a lesser known and lesser explored use of VR. However, many fine artists ms can be visited virtually through VR technology.

Urban design

litate a design. Urban design is also used to find loopholes and faults in a design. Urban ding, transport planning, landscape design, and so on.

Motion pictures

We can find some motion pictures that use the concept of VR technology. The motion picture *Avatar* is a great example of this. The whole concept lies within VR technology. The concept portrays a virtual life and activity with the help of technology so that the character can experience a virtual world without being present there in reality.

on.

Today, spectator experience has been increased through VR.

Medical therapy

Virtual reality therapy (VRT) is quite popular in medical science, especially in greater than 90% worldwide.

VRT is used mainly to treat people who fear height, flying, insects, motion, public talking, and so on, to create a controlled virtual environment.

VR in Android games

VR in
iously,

This SDK is still out there in the market and is being used widely.

. The
trean
eam very
soon as Android has now included a special setting for VR.

History of Android VR games

In the late 70s, the VR system started evolving at a rapid speed with better DK in 2016.

The latest Android devices are VR capable. It is believed that most of the upcoming devices will support VR headsets.

. But
ts are
considered a part of Android wearable devices having a high configuration.

Technical specifications

However,
with improving technology, the latency is decreasing and the experience is getting VR, and
so on.

droid

ver,

now, Android devices can be directly associated with VR headsets with multiple VR headsets

are only compatible with defined mobile handsets due to their own physical size and hardware specifications. VR headsets are designed to fit certain screen sizes. However, there are a few VR kits available in the market that support multiple screen sizes.

Current Android VR game industry

handsets

1 VR kits

specially designed for a particular handset.

Oculus VR

sets. However, with the help of Android VR development, most mobile devices are capable of running a VR application and they can be experienced with an external VR kit.

Future of Android in VR

oid is

of VR

support. This clearly shows that Android has an immense potential for VR games. Google is potentially working on a future VR-specific platform.

There are more devices coming to market that are VR compatible. So, there is a bright future for VR games on the Android platform.

Google Daydream

id to be

the successor to Google Cardboard. The latest Android N will include support for nced as

"Daydream-ready phones".

hts

in the digital gaming world. The experience and the quality of the game is going to be better, smoother, and more realistic.

Game development for VR devices

There is a large space for VR in the mobile game industry. Based on Android and iOS, VR game development is different from other mobile games. Certainly, game design and planning are also different from other mobile games.

VR game design

VR does not fulfil the criteria for every genre. Hence, VR game design needs to be done for most genres, preferably a first person shooter or in some RPG or racing games.

Experience is an important factor for any VR games.

Generally, a game designer starts designing a game from an idea. Then, corresponding controls, environment, and experience are thought of. However, in the case of VR games, the developer or designer already has a defined set of features on which they execute an idea through design.

VR target audience

One of them is that more and more devices are being launched with VR capability.

It requires a supported VR headset too. Not every user will go and buy extra equipment to play VR games. That is why casual players are not the main target audience for VR games at the moment.

The use of VR is vast. A major section of the use is simulation, which includes education. Gamification of the education process opens up a huge target audience and curious about new things.

VR game development constraints

should be aware and well-versed in Android, and efficient enough to understand VR specifications and platform limitations. Here are a few constraints when developing games for VR on the Android platform:

- Limited handsets to support
- Limited and specific target audience
- Limited controls
- Limited graphical quality with maximum experience

Introduction to the Cardboard SDK

14 for use with a head-mounted device for a smartphone. This platform targets a low-cost project to encourage VR application development on a massive scale, which has proven to be fruitful till date. Google declared Daydream to be the next step for this platform on May 18, 2016.

The name "Cardboard" came from the concept of a VR device made with cardboard, which makes the device significantly cheaper. However, many third-party ls to increase its style and build quality.

or Android and iOS. This has changed the VR development concept, which was limited to typical device and hardware specifications:



Cardboard headset components

A typical Google Cardboard headset contains the following parts:

- A piece of cardboard cut into a precise shape
- 45 mm focal length lenses
- Magnets or capacitive tape
- A hook and loop fastener
- A rubber band
- An optional near-field communication tag

Each part of the cardboard device is either pre-fitted or has a mechanical slot to fit in. It is easy and fast to assemble the full gear. The rubber band is fitted last to wear the purpose of a VR experience.

device slot of the VR headset and held in place by the corresponding components.

Cardboard application working principle

to two, es barrel the lenses.

Thus, a complete wide 3D world is created.

Initial Cardboard headsets could fit phones with screens up to 5.7 inches (140 mm) and used magnets as input buttons, which also required a compass sensor in the

Upgrades and variations

2016, ed the input button with a conductive lever that triggers a touch event on the smartphone's screen for better compatibility across devices.

Google allows several vendors and manufacturers to build Cardboard-compatible t of variants of this product.

Basic guide to develop games with the Cardboard SDK

similar to other Android games. Let's have a quick look at the basics of Cardboard development styles and standards through these points:

- Launching and exiting the VR game
- VR device adaptation
- Display properties
- In-game components
- Game controls
- Game audio setup
- User focus assistance
- Ultimate VR experience

Launching and exiting the VR game

asks and

it takes time

to mount the Android device to a VR headset properly, so the developer does not
d wait for

the user to start it after it is in the perfect situation for running.

VR sign

or button to start the VR game.

There are two possible exits for a standard VR game:

- Hitting the Back button
- Hitting the Home button

Hitting the Back button

If the VR has a 2D interface to show popups for exiting, then using the Back button
f hitting

the back button accidentally as the device is mounted in the VR headset. It is very
common to use a single hit on the back button to exit the game, because of the
larly to a
non-VR game.

Hitting the Home button

Generally, hitting the Home button pushes the Android application to the background without killing the application, in the case of a VR game.

VR device adaptation

A VR game for Android using the Cardboard SDK should adapt the physical characteristics of the VR headset. The Google Cardboard SDK has a feature to carry out this job automatically. The developer can rely on the SDK to adjust application settings and configuration according to the VR headset. The Cardboard SDK itself contains adjustment settings for few specific Cardboard devices. The SDK can configure stereo settings and correct distortion for few specific lenses of a VR headset.

It is recommended that the developers use the Cardboard SDK feature to support the best possible VR experience without any hassle.

Display properties

In many Android devices, there is a feature called Lights Out. In those devices, the Home, Menu, and Back controls are hidden under the Lights Out feature. VR games set lenses.

mode.

eral vision,

blocking or distracting them from the actual VR experience.

In-game components

Normally, the VR game experience lies within the environment through the device screen. It is very unlikely to trigger any popups and other unwanted components on the screen during gameplay.

Developers must not call any API that will trigger any popup or any unwanted

component rendering. Forcefully rendered 2D elements may cause disorientation of
ut from the VR

ch is not at

all convenient.

Game controls

n in VR
trol scheme
of VR games.

Control concepts

d place
the UI controls in the initial field of view so that users can locate them to start the
gameplay. If the controls are not in the visible range, then users might get confused.
e game. In
both cases, users may lose interest in the game.

While playing the game, if there is any user interface, those controls need to move
along with the field of view. Otherwise, users might have to go back to the place
where the UI element was.

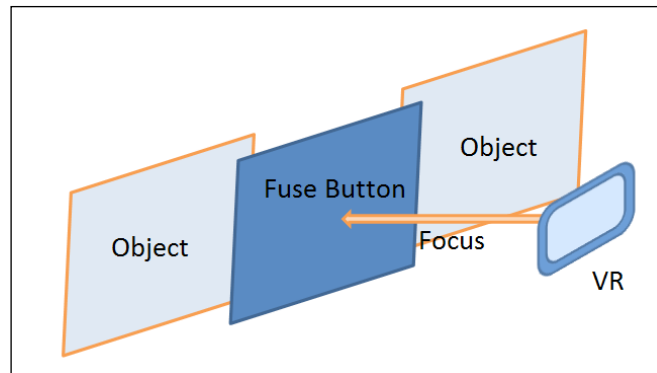
Types of controls

veral ways
s of controls
are as follows:

- Fuse button
- Visual countdown

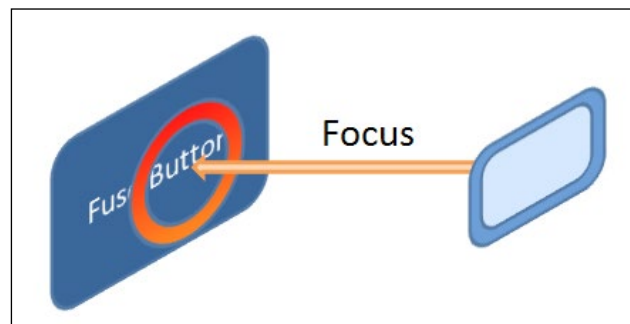
Fuse button

can be used
to click on targets. One of the uses of this button is to trigger a virtual button in the
VR world which will fuse. This means a corresponding task will trigger after some
time of focusing on the virtual fusing button. However, it may be frustrating, as the
e developer
er possible:



Visual countdown

user might focus on the button unknowingly, and after a certain amount of time, it changes the continue the same game experience. The developer should indicate the countdown visually to of time:



Fuse button indication

We already know that 2D UI buttons are not supported by VR games. So, developers There must . Then, the action can be performed on countdown or by click.

For example, developers often use some glow, shine, shake, or other dynamic r action is defined by the surroundings to indicate the task.

Control placement

It is large enough to focus on. It can save the user a lot of confusion. Moreover, there should not be adjacent fuse buttons in the scene that are difficult to locate and might create confusion.

VR game development through Google VR

featuring both Android SDK and Android NDK. These SDKs support both the Cardboard and Daydream VR platforms.

skills:

- Head-tracking system
- Spatial audio
- Dynamic rendering
- UI handling
- 3D calibration
- Lens distortion correction
- Stereo geometry configuration

Let's have a look at the Android SDKs for VR game development:

- Google VR using the Android SDK
- Google VR using the Android NDK

Google VR using the Android SDK

Android can be used independently or with Android Studio.

The developer can use other tools instead of Android Studio, but it is strongly recommended to use Android Studio for Android builds. It is the most convenient method for VR application development on the Android platform.

ate a VR
application build for Android:

- Android Studio version 1.0 or higher
- Android SDK version 23 or higher
- Gradle version 23.0.1 or higher

Using Android Studio can save users from configuring Gradle settings to build
adde if
required.

The developer can choose Gradle to build an Android application project. In that
case, the developer needs to manually edit each `build.gradle` file of every module
to include `.AAR` declarations for the Gradle build.

The modification has to be done this way:

```
dependencies
{
    compile(name:'audio', ext:'aar')
    compile(name:'common', ext:'aar')
    compile(name:'core', ext:'aar')
}

repositories
{
    flatDir
    {
        dirs 'libs'
    }
}
```

Android Studio automatically makes these changes and declares dependencies
for each module. This will tell Gradle to look in the `libs` subdirectory of the
corresponding module for the three `.AAR` declarations. If there is no subdirectory
called `libs`, then the developer needs to create a `libs` subdirectory inside the
module's directory and copy the required `.AAR` files manually.

Google VR using Android NDK

K
development. It requires the following components:

- Android Studio version 1.0 or higher
- Android NDK
- Google VR SDK for Android

It is recommended that you use the Android NDK with the Daydream development for the Daydream SDK.

rsion
-tracking
system to power up the development.

Android SDK is sufficient to develop VR games with Cardboard, but a few developers like to use native languages such as C++ to develop games. Having a better understanding of technology, a few choose to develop VR games with C++ and OpenGL. In this way, the VR game can be portable to other VR platforms as well.

Android VR development best practices

Developers need to have experience of coding regular games for Android before they start building a VR experience with Cardboard. Here are a few areas that developers need to keep an eye on while developing VR games for Google Cardboard:

- Draw call limitations
- Triangle count limitations
- Keeping a steady FPS
- Overcoming overheating problems
- Aiming for a better audio experience
- Setting up proper project settings
- Using a proper test environment

Draw call limitations

is always
nd reduce
GPU overhead.

developer
industry,
100.

Triangle count limitations

3D games.

The same logic applies to VR games. However, it is more difficult to maintain
ly use
triangle counts within 100k.

Currently, the average triangle count of a decently performing VR game is around
ze them in the
minimum possible triangles and vertices to achieve a decent FPS at runtime.

Keeping a steady FPS

ames

s, the

developer can reduce rendering time to gain performance.

So, the

developer should decide and use the minimum possible game objects or elements to
keep the FPS steady.

ith limited

resources. Designing and creating art assets is a major part. Using low-poly models
ty can be

improved with the help of excellent texturing support for the model and strategic
light mapping in the virtual reality world.

Overcoming overheating problems

Overheating is a common problem for Android VR games. The device heats up while
running a VR application due to the extensive use of CPU and GPU. There is not
much a developer can do to overcome the problem completely. However, developers
can optimize the VR game to reduce processor use. The game should have limited

Better audio experience

Android VR games are played with the support of extra VR devices, such as Google Cardboard or similar. We have already discussed the devices in this chapter. Such devices do not have audio support; VR games use device speakers by default.

erience.

uggest

that users wear headsets while playing the VR game.

Setting up proper project settings

Setting VR project settings right early on can save the developer from headaches in the future. To get better project performance in particular, it is very important to set the quality settings properly at the beginning of the project. The entire project and performance planning cannot be fruitful without a prior project configuration.

Using a proper test environment

of any game,

especially for large-scale games such as VR. The developer must know the status the game

better. In this way, the development process can run smoothly.

de the VR

device. It is also recommended that you use the **adb** to check each debug condition and statement. It is very difficult to use a normal debug bridge for VR games as the reless

to run

n save a

lot of time, which can be used to improve the game.

Challenges with the Android VR game market

mes on

t a developer

might face while developing or monetizing VR games for Android:

- Low target audience
- Limited game genres

- Long game sessions
- Limited device support
- Real-time constraints

Low target audience

There are very few Android users who are familiar with the VR concept and technology. Mostly, users are the common handset holders who do not have VR headsets. So, there is already a major section of the audience who are out of scope of VR headsets.

Limited game genres

Android VR games cannot do so much in VR gaming market in securing a profitable position and design monetization aspects.

Long game sessions

Android VR games must be played with a VR headset fitted with an Android OS and a suitable environment to set up a playable environment.

Android game users are usually addicted to quick and flexible game sessions. Most of the games can be immediately paused and resumed for convenient gameplay. However, VR games cannot be paused or resumed immediately. Users prefer to play VR games when they have a long period of free time.

Limited device support

Android VR games require high-level hardware configuration to run the game. The number of devices supporting VR is limited, thus limiting the possible time.

There are many cheap and low-configuration handsets available in the market. Millions of Android users use those handsets. However, most of them are incapable of running VR games. This limitation reduces the number of supported devices significantly.

Real-time constraints

y also
must
ruptions.

avoid any
dset, so

users need to take these safety measures. These issues cannot be resolved and this affects the VR gaming market and leads to low session count and time.

Expanded VR gaming concepts and development

se ideas
is a
broader concept of virtual gaming.

gaming, such
as *Paint Ball*, *Laser Tag*, and so on. With the help of VR technology, these experiences can reach the next level. Using a VR headset in a predefined arena that is physically synced with the VR application environment can take the user into the game.

as for VR

al gears

and sensors. These gears and sensors track the user's movement in real time and duplicate it within the virtual world through the VR headset display.

tual world,
nd

researchers are making it better day by day. However, the Android platform is yet to take a step into this kind of VR development for the following reasons:

- Android is a mobile OS that works best on portable devices.
- Most Android devices are mobile handsets or tablets.
- It is difficult to manage a dedicated hardware platform with a large physical setup through Android. **Real-time operating systems (RTOS)** can perform much better than Android for such systems.
- There are low market prospects for such Android setups, and the costing may be huge.

ge scale,

it is believed that Android will become part of such systems soon.

However,
VR feature.
eam,
which includes extended controls with a separate controller.

- [253]

11

Android Game Development Using C++ and OpenGL

an

Android game. The Android SDK is very capable of taking care of both. Certainly, development toolset

in native languages such as C and C++?" Compared to Java, C and C++ are much more difficult to manage and write code in. The answer lies in the question itself. The Java architecture runs on JVM, which is associated with the Android operating system. This creates an extra latency, which has a performance lag. The scale of the lag depends on the scale of the application.

A highly CPU-intensive application may cause a significant amount of visible lag in

r, native

not

possible in the case of the Java architecture used by the Android SDK.

nGL

can also choose Android as the target platform. Native code helps directly structure the application using OpenGL.

e following

topics:

- Introduction to the 3Android NDK
- C++ for games – pros and cons
- Native code performance
- Introduction to OpenGL
- Rendering using OpenGL
- Different CPU architecture support

Introduction to the Android NDK

Android
his is
where the Android NDK comes into the picture.

The Android NDK is a toolset used to develop a module of an application that will
++ have
the ability to interact with a native component directly, which reduces the latency
between the application and hardware.

How the NDK works

An Android native code segment interacts with main application through the **Java Native Interface (JNI)**. The Android NDK comes with a build script that converts

ndroid
application as per requirement. An NDK build script creates `.so` files and adds them
to the application path.

The Android build process creates Dalvik Executable files (`.dex`) to run on the
recognizes
are
declared with the `native` keyword:

```
public native void testFunc (int param);
```

The method always has public access, because the native library is always treated as
an external source. Here, the developer should always keep in mind that there should
never be multiple definitions of a method for the same declaration collectively for all
the included native libraries. This will always create a compilation error.

of libraries:

- Native shared library
- Native static library

Native shared library

The native build script creates `.so` files from C++ files, which is termed as the native
the true
sense. An Android application is a Java application, but a native application can be
triggered through a native shared library.

For game development, if the game is written in a native language, then the game code is included in the shared library.

Native static library

d represented

by .a files. These libraries are included in other libraries. A compiler can remove unused code during compilation.

Build dependency

project

into an APK file with the support of Java. However, the NDK is not sufficient to build and package APK files. Here are the dependencies for creating an Android application APK other than the NDK:

- Android SDK
- C++ compiler
- Python
- Gradle
- Cygwin
- Java

Android SDK

ute necessary

to have Android SDK in order to create an Android application package.

C++ compiler

uired to

compile the code base on the development platform. C++ compilers are platform dependent, so it may not be the same C++ compiler on each platform.

For example, on a Windows machine, the C++11 compiler is used currently in the development industry, whereas the GC++ compiler is used on Linux machines.

These may create different code bases for the actual development project in terms of syntax and API calls.

Python

ications

for Android and can support multiple platforms by converting the source into
for the
conversion of C++ code to native binary.

Gradle

convert

ment to make

application packages.

Cygwin

The Windows system does not have a Unix environment. Cygwin is required to
provide a virtual Unix environment to support the building platform.

Java

cation

pment.

Native project build configuration

An Android project needs the following configurations in order to create an
nds on the
configuration defined in these two files:

- `Android.mk`
- `Application.mk`

Android.mk configuration

Location

The `Android.mk` file can be located at `<Application Project Path>/jni/`.

Configuration options:

The `Android.mk` file contains the following options to create an application package:

- `CLEAR_VARS`: This clears the local and user-defined variables. This option is
invoked by the `include $(CLEAR_VARS)` syntax.

- `BUILD_SHARED_LIBRARY`: This includes all local files, defined in `LOCAL_MODULE` and `LOCAL_SRC_FILES`, in a shared library. It is invoked by the `include $(BUILD_SHARED_LIBRARY)` syntax.
- `BUILD_STATIC_LIBRARY`: This specifies static libraries to create .a files used by the shared libraries. It is invoked by the `include $(BUILD_STATIC_LIBRARY)` syntax.
- `PREBUILT_SHARED_LIBRARY`: This indicates a prebuilt shared library at a path. It is invoked by the `include $(PREBUILT_SHARED_LIBRARY)` syntax.
- `PREBUILT_STATIC_LIBRARY`: This indicates a prebuilt static library at a path. It is invoked by the `include $(PREBUILT_STATIC_LIBRARY)` syntax.
- `TARGET_ARCH`: This indicates the basic type of processor architecture family such as ARM, x86, and so on.
- `TARGET_PLATFORM`: This defines the target Android platform. The mentioned

SDK manager. It indicates the Android API level in order to create the application package.

- `TARGET_ARCH_ABI`: This indicates the specific ABI for target processor architecture, such as armeabi, armeabi-v7, x86, and so on.
- `LOCAL_PATH`: This points to the current file directory. This variable does not get cleared by the `CLEAR_VARS` command. It is invoked by the `LOCAL_PATH := $(call my-dir)` syntax.
- `LOCAL_MODULE`: This indicates all the unique local module names. It is invoked by the `LOCAL_MODULE := "<module name>"` syntax.
- `LOCAL_MODULE_FILENAME`: This indicates the library name that contains the defined `LOCAL_MODULE`. It is invoked by the `LOCAL_MODULE_FILENAME := "<module library file name>"` syntax.
- `LOCAL_SRC_FILES`: This indicates all the native source code file paths to be compiled into a shared library. It is invoked by the `LOCAL_SRC_FILES := <Local source file path>` syntax.

There are other optional configurations that can be set in this file, such as `LOCAL_C_INCLUDES`, `LOCAL_CFLAGS`, `LOCAL_CPP_EXTENSION`, `LOCAL_CPP_FEATURES`, `LOCAL_SHARED_LIBRARIES`, `LOCAL_STATIC_LIBRARIES`, and `LOCAL_EXPORT_CFLAGS`.

Application.mk configuration

Location

The Application.mk file can be located at <Application Project Path>/jni/.

Configuration options

The Application.mk file contains the following options to create an application package:

- APP_PROJECT_PATH: This is the absolute path to the project root directory.
- APP_OPTIM: This indicates the optional setting to create the build package as release or debug.
- APP_CFLAGS: This defines a set of C-compiler flags for the build instead of changing in the Android.mk file.
- APP_CPPFLAGS: This defines a set of C++-compiler flags for the build instead of changing in the Android.mk file.
- APP_BUILD_SCRIPT: This is an optional setting to specify a build script other than the default jni/Android.mk script.
- APP_ABI: This option specifies the set of ABIs to be optimized for the

each ABI support:

- ARMv5: armeabi
 - ARMv7: armeabi-v7a
 - ARMv8: arm64-v8a
 - Intel 32-bit: x86
 - Intel 64-bit: x86_64
 - MIPS 32-bit: mips
 - MIPS 64-bit: mips64
 - ALL-SET: all
- APP_PLATFORM: This option specifies the target Android platform.
 - NDK_TOOLCHAIN_VERSION: This option specifies the version of the GCC compiler. By default, versions 4.9 and 4.8 are used for compilation in 64 bit and 32 bit, respectively.
 - APP_STL: This is an optional configuration to link alternative C++ implementations.
 - APP_LDFLAGS: In the case of building a shared library and executables, this

C++ for games – pros and cons

t go

f game

development. C++ has a slight performance edge over Java, and Java is known for its simplicity.

There may be many programmers who are more comfortable in C++ than Java, or vice versa. In game development, personal choice of programming language does e

requirements. It is always recommended that you use the Android SDK to develop an application rather than using the NDK.

Let's discuss the advantages and disadvantages of using native language for game programming.

Advantages of using C++

Let's first have a look at the positive side of using C++ for game programming through the following points:

- Universal game programming language
- Cross platform portability
- Faster execution
- CPU architecture support

Universal game programming language

In the case of game development, C++ is widely used for many platforms, especially for consoles and PC game development. This is the reason many game engines opted for C++ as the primary programming language.

Sometimes, it is difficult to learn many programming languages to work on different platforms with different architecture. C++ provides a most common solution to this I use.

Cross-platform portability

The same C++ code is compiled into a library targeting a specific operating platform. Thus, the same project can be compiled for different platforms. Hence, it is super easy to port a game to various platforms if it is written in C++.

For example, the famous and effective cross-platform game engine Cocos2d-x uses or many platforms such as Android, iOS, Mac OS, and Windows.

Faster execution

ames
he

CPU architecture support

C++ code can be compiled for specific target CPU architectures such as x86, ARM, Neon, or MIPS. This specification indicates better performance on that particular processor.

Compiler configuration for CPU architecture in Android NDK ensures the best define each and every platform to avoid extra compilation.

Disadvantages of using C++

Now, let's discuss the other side of the coin through these points:

- High program complexity
- Platform-dependent compiler
- Manual memory management

High program complexity

,
s in
r to take
care of every programming aspect.

C++ itself has a complex architecture compared to Java. The chances of facing exceptions and errors increases if C++ is used.

Platform-dependent compiler

Going cross platform is easy when using C++. However, configuring the build script can be a pain in most of the cases. It is a very common scenario that the same game fails to run on a ported platform due to the wrong configuration. Moreover, it becomes difficult to find out the issue as the game is successfully running on some other platform.

Most of the time, different platforms use different C++ compilers. So, it requires an extra effort to identify platform-specific code and find out an alternative for each platform, if required.

Manual memory management

Java does not require memory management to be implemented by the developer, and the memory is efficiently managed by JVM (DVM in the case of Android). So, the garbage collector can cause a severe drop in performance.

The developer should use optimum memory, because the garbage collector cannot free it.

Conclusion

C++ has its own advantages. However, when it comes to game programming for Android, the amount of effort and risk taken by opting for C++ than coding in Java for Android, Java should always be preferred for Android. DVM runs Java code efficiently enough to achieve reasonable performance on Android devices. Moreover, the Android NDK is a better option. Even when DVM and a native application written in C++, it does not help much.

If the developer chooses not to go for cross platform and keeps the game scope within Android only, then it is recommended that you use Android SDK rather than Android NDK. It will decrease the development hassle and complexity with a negligible amount of performance loss.

Native code performance

As we already know, native code can run faster with better processing speed. This can be further optimized for a specific CPU architecture. The main reason behind this performance boost is the use of pointers in memory operations. However, it depends on the developer and the coding style.

of
performance gain in native language.

Consider this Java code:

```
int[] testArray = new int[1000];
for ( int i = 0; i < 1000; ++ i)
{
    testArray[i] = i;
}
```

In this case, the address of 1000 fields in the array is handled by JVM (DVM in the case of an Android Dalvik system). So, the interpreter parses to the i^{th} position and performs an assignment operation each time, which takes a lot of time.

and
use pointers:

```
int testArray[1000];
int *ptrArray = testArray;
for ( int i = 0; i < 1000; ++ i)
{
    *ptrArray = i;
    ptrArray += i * sizeof(int);
}
```

In this example, the interpreter does not need to parse to the target memory location. The address of the location is pointed out by `ptrArray`. Hence, the value can be directly assigned to the memory location.

Especially for multi-dimensional arrays, a significant performance gain can be onality.
rocessing,
where a huge amount of data is processed at a time.

Rendering using OpenGL

Android uses OpenGL for rendering. Android SDK libraries include the OpenGL libraries, specially optimized for Android. Android started supporting OpenGL from API level 4 and then increased its support as the level increased. Currently, the maximum supported version of OpenGL is OpenGL ES 3.1 from API level 21.

OpenGL versions

and 2.0 have a lot of differences in terms of coding style, API convenience, functionality, and significant to Android development:

- OpenGL ES 1.x
- OpenGL ES 2.0
- OpenGL ES 3.0
- OpenGL ES 3.1

OpenGL 1.x

are the OpenGL ES 1.x library, `libGLESv1.so`. The headers `gl.h` and `glxext.h` contain all the necessary APIs for OpenGL functionality.

OpenGL 2.0

are used, because and fragment shaders useful for games. OpenGL ES 2.0 can be used in Android native development projects by including the `libGLESv2.so` shared library in the project, as follows:

```
LOCAL_LDLIBS := -lGLESv2
```

The headers are `gl2.h` and `gl2ext.h`. OpenGL ES 2.0 is supported from Android API level 5.

OpenGL 3.0

include
libGLESv3.so to use OpenGL 3.0, as follows:

```
LOCAL_LDLIBS := -lGLESv3
```

The headers are `gl3.h` and `gl3ext.h`.

OpenGL 3.1

include
libGLESv3.so to use OpenGL 3.1, as follows:

```
LOCAL_LDLIBS := -lGLESv3
```

The headers are `gl31.h` and `gl3ext.h`.

s. If a
eck before
using the version. Also, proper version of OpenGL ES must be used to run the game
S 3.2.

Detecting and setting the OpenGL version

This piece of Android Java code can be used to implement proper OpenGL ES support for an Android game:

```
private GLSurfaceView glSurfaceView;  
void setOpenGLVersion()  
{  
    final boolean supportOpenGLES3 =  
        configurationInfo.reqGLESVersion >= 0x30000;  
  
    if (supportOpenGLES3)  
    {  
        glSurfaceView = new GLSurfaceView(this);  
        glSurfaceView.setEGLContextClientVersion(3);  
        glSurfaceView.setRenderer(new RendererWrapper());  
        setContentView(glSurfaceView);  
    }  
    else  
    {  
        final boolean supportOpenGLES2 =  
            configurationInfo.reqGLESVersion >= 0x20000;  
  
        if (supportOpenGLES2)
```

```
    {
        glSurfaceView = new GLSurfaceView(this);
        glSurfaceView.setEGLContextClientVersion(2);
        glSurfaceView.setRenderer(new RendererWrapper());
        setContentView(glSurfaceView);
    }
    else
    {
        glSurfaceView = new GLSurfaceView(this);
        glSurfaceView.setEGLContextClientVersion(1);
        glSurfaceView.setRenderer(new RendererWrapper());
        setContentView(glSurfaceView);
    }
}
```

Texture compression and OpenGL

Texture compression has a significant effect on the rendering process handled by OpenGL. It can increase or decrease performance for different types of texture compression. Let's have a quick look at some of the important texture compression formats:

- ATC
- PVRTC
- DXTC

ATC

ATI texture compression is often called ATITC. This compression supports RGB with and without an alpha channel. This is the most common and widely used compression technique for Android.

PVRTC

Power VR texture compression uses 2-bit and 4-bit pixel compression with or without a global value.

DXTC

DXTC uses a 4-bit or 8-bit ARGB channel.

OpenGL manifest configuration

Android requires the version definition of OpenGL used in the application, along with other required options.

Here is the version declaration syntax for OpenGL ES:

```
<uses-feature android:glEsVersion=<Target version goes here>
    android:required="true" />
```

Here are the target version options:

- 0x00010000 for version 1.0
- 0x00010001 for version 1.1
- 0x00020000 for version 2.0
- 0x00030000 for version 3.0
- 0x00030001 for version 3.1
- 0x00030002 for version 3.2

Here is the optional setting for texture compression declaration:

```
<supports-gl-texture android:name=<Compression support type goes
    here> />
```

These are the compression type options:

- GL_OES_compressed_ETC1_RGB8_texture
- GL_OES_compressed_paletted_texture
- GL_EXT_texture_compression_s3tc
- GL_IMG_texture_compression_pvrtc
- GL_EXT_texture_compression_dxt1
- GL_EXT_texture_compression_dxt2
- GL_EXT_texture_compression_dxt3
- GL_EXT_texture_compression_dxt4
- GL_EXT_texture_compression_dxt5
- GL_AMD_compressed_3DC_texture
- GL_EXT_texture_compression_latc
- GL_AMD_compressed_ATC_texture
- GL_ATI_texture_compression_atitc

developer
 aware
 and Android version requirement.



Google does the filtration process of devices automatically if the target device does not support the declared texture format or formats.

Choosing the target OpenGL ES version

. So, it is
 e game.
 L version:

- Performance
- Texture support
- Device support
- Rendering feature
- Programming comfort

Performance

which is
 way faster than OpenGL 1.x. So, it is always better to use the latest possible version
 in the game.

Texture support

Texture compression support varies with OpenGL versions. Older versions support
 older texture compression factors. Also, Android version support is not universal
 version for
 texture support.

Device support

ons of
 OpenGL are not supported by all devices. So, in order to target a bigger range of
 devices, the user should change the OpenGL version to 2.0 as most devices support
 this version.

Rendering feature

t factor
equired
sion.

Programming comfort

. The
developer should choose the version if it can actually be developed in the company
with ease.

Different CPU architecture support

a separate
ture. However,
this feature comes at a significant cost. Let's have a look at the details of this feature.

Available CPU architectures

Here are the architectures currently supported by the NDK build:

- ARM
- x86
- Neon
- MIPS

ARM

ARM stands for **Acorn RISC Machine**. This is a **RISC (Reduced Instruction Set Computing)** based processor, mainly targeting embedded or mobile computing. As the base says, it is highly efficient for an operating system such as Android.

Currently, most used processors of the Android platform are from the ARM family. It can be further sub-categorized as follows:

- ARMv5TE
- ARMv7
- ARMv8

x86

Intel introduced the **x86** architecture for processors. At first, these processors were used in mobile devices in the form of Celeron or Atom processors.

Two types of x86 architecture can be set for the Android NDK build:

- i686
- x86-64

Neon

The **Neon**le computation. The Android build also can be optimized for this specific architecture. All Cortex processors are basically Neon-based processors.

MIPS

MIPS stands for **Microprocessor without Interlocked Pipeline Stages**. There is says, this architecture is used in microprocessors in embedded devices for small-scale ure. However, this type of processor is rarely used in Android systems today.

Advantages and disadvantages of integrating multiple architecture support

Android mobile devices have different configurations in terms of memory and e the performance that comes with greater build size.

rocessor and includes it in the build package.

Here are some advantages and disadvantages of providing separate processor architecture support.

Let's see the advantages first:

- **Faster operation:** Separate architecture for a separate processor results in a faster processing speed of game instructions. If the processor architecture is o perform any conversions and can run the instructions at a faster speed.

- **Optimum use of processor:** The operating system always looks for the specific architecture for an integrated processor. The same architecture makes optimum use of the processor.
- **Minimum power consumption:** Optimum processing directly implies optimum and minimum power usage for processing.
- **Optimum memory usage:** The processor does not need to use extra runtime memory to execute instructions if the same processor architecture is supported by the Android application.

Let's see the disadvantages now:

- **Larger build size:** Using a separate shared library for a separate architecture optimization.
- **Reduced target device count:** If the size of the APK is large, it creates more problems to accommodate it for a low storage device. Hence, device support becomes less.

Summary

We looked at Android NDK briefly in this chapter and cleared a few doubts on games not always true. Processing and performance depend on the development style and standard. In most common scenarios, the difference between native development and SDK development is negligible.

depend on aspects and what we should use. Clearly, OpenGL ES 2.0 is a good choice as most Android OpenGL ES 3.0 is not supported by most Android devices yet.

Until now, we have covered almost every aspect of Android game development. However, finishing the implementation for the game does not define the completion. It is just a polishing in the end.

12

Polishing Android Games

The quality of a developed game mostly depends on the final polishing. Polishing possible aspect to provide maximum user experience. There is no limit to such improvisation. Most game developers allocate a major time period to polishing.

e time, the developer faces a time crunch at the end of the development process. Polishing takes a significant user experience and retention point of view.

designers, and producer to ensure the target polishing level of the game.

Many developers choose to carry out play testing with a significant but limited number of users. Then, the issues and improvements are charted down for polishing. There are several approaches to polishing an Android game used by developers. We will discuss the general and widely used methods and practices of polishing in this chapter.

We will have a detailed look at the following topics:

- Requirements for polishing
- Play testing
- Taking care of UX
- Android-specific polishing
- Game portability

Requirements for polishing

Polishing any game defines the quality of development. So, it is absolutely necessary for developers to have a good understanding of the requirements for polishing their game to provide a better user experience.

Polishing Android games covers all the three development components of a game:

- Development polishing
- Art polishing
- Design polishing

Development polishing

Development polishing is a process to optimize the game's performance and reduce the size of the APK file.

This section includes programming optimization, memory optimization, and stripping unnecessary code blocks to avoid any extra processing.

Development polishing can be further split into three phases:

- Memory optimization
- Performance optimization
- Portability

Memory optimization

Memory optimization is a process to reduce the memory usage of the game.

. In

memory optimization, it helps a lot to increase device support and game stability. A good game must be able to run on devices with limited memory capacity.

Performance optimization

Performance optimization is a process to improve the game's performance.

Every developer wants their game to run smoothly on every target Android device. However, it is not always possible to test such smoothness in all devices. Mostly, developers select a few devices that are almost equivalent to other targeting devices to test the game.

Portability

development

minimal

effort. Portability might be the key to success for many Android games.

Art polishing

game art is

to provide better visual quality within the same art space.

game art may

Android, where

rent visual

quality, game art polishing becomes extremely useful.

There are mainly three phases of art polishing:

- UI polishing
- Animation polishing
- Marketing graphics

UI polishing

UI drives the game flow. So, the UI art should convey the desired path easily for

any to

polish the UI art accordingly.

Animation polishing

ions

means increasing the visual effectiveness and make a user see the game from a developer's point of view. Mainly for sports games, FPSs, and RPGs, animations are inevitable. Animations decide the character of gameplay.

Marketing graphics

Marketing assets are the first thing to be visualized when it comes to a game. They create the hype and interest to start playing the game for the user. If marketing art is not polished enough to attract users to the game, then there may be significant loss, irrespective of the actual game quality.

Design polishing

can be the design after development so that the final application can have improved quality. It has five phases:

- Designing UX
- Polishing the game flow
- Polishing the metagame
- Game economy balance
- Game difficulty balance

Designing UX

r's point of view. There are several cases where a game failed to retain users because of poor UX or.

Polishing the game flow

Often in the game development process, the game flow might contain some unnecessary loops or actions. Users should have the maximum experience of the game with minimal action. Each action should be simplified enough for the users to simplify the game flow to that level. But it should be simplified enough to make it easy to understand.

Polishing the metagame

Polishing the metagame means polishing the packaging so that the game becomes easy for success in terms of revenue.

Game economy balance

ishing, depending on the core game model. Almost every game has an economical aspect associated to it and give them a sense of progression.

Game difficulty balance

As they say, all the fingers on a hand are never the same. Similarly, user efficiency is also not the same. It is the most likely thing to vary, and is reflected on the game leaderboard. So, the difficulty of the game should be balanced in a way such that almost each and every player has a chance to keep playing the game.

Play testing

ed out after the
the entire
user behavior throughout the game.

Here are the fields of exploration during play testing:

- User gameplay difficulty level
- User actions during gameplay
- User actions while browsing the game
- Whether the user is paying or not
- Whether the game is running smoothly
- Whether the user can adopt the gameplay
- User retention

elopers
release beta versions of the game in a certain region to carry out play testing. The
disadvantage
in the play
fter play
e full game
before
performing play testing.

User gameplay difficulty levels

Difficulty
the same game are not equally efficient in playing the game. Play testing reveals the
difficulty faced by users while playing the game.

sting result.
This has a direct impact on game polishing.

User actions during gameplay

For example, such as swiping, tapping on different buttons, choosing options, and so on. The developer needs to consider the details of each action that may be considered.

Game control.

Essence of the

game. Sometimes, developers change the game control if they encounter a serious issue with regard to user actions.

User actions while browsing the game

During

the development phase. The UI flow and navigation style of the game are validated throughout this process. Sometimes, a UI section may be overlooked by the user. It is very difficult for developers to identify such UI sections from a user's point of view, although developers can easily browse those segments as they themselves have implemented those UI sections. Such cases indicate that the section of UI that is overlooked by a decent number of users is not highlighted enough by any means.

Outside the main game

flow, such as the leaderboard, offer wall, achievements, help, settings, IAP screens, secondary game mode, and so on. If a user does not visit such UI sections for a long time, the developer

may choose to change the UI style or find out an alternate solution. The success of metagames mostly depends on this kind of polishing. Game monetization can also be improved a lot.

Whether the user is paying or not

There are several game monetization models available. The basic three types are premium, free, and freemium. Developers adopt any model for the game to generate revenue.

As the name suggests, premium games are basically paid games. This means the full game is bought by the user in the first instance. So, in this case the user does not need to pay while playing. A free game is completely free to play and has no provision for paying to gain any advantage while playing. The developer can plan revenue through game advertising. User actions and behavior during gameplay can help place advertisements strategically. Users have an option to pay after starting to play the game in the case of the freemium model. The developer designs the metagame to make users pay for the game to gain advantage or increase game progression speed.

In the play testing stage, the developer monitors users when they are paying for the game. In the freemium model, the developer defines stages where the user should pay to progress faster or more smoothly. This plan is validated through play testing to project future revenue.

Whether the game is running smoothly

As we have already discussed previously, from the optimization point of view, smooth gameplay is one of the major segments of game polishing. Initial testing is carried out on a few restricted devices. However, in the case of play testing, it is much more reliable to focus on real-time scenarios with a real device to validate smooth gameplay. However, a variety of hardware configurations are available on Android. The developer must decide the test configuration and set the benchmark before play testing.

ance
data through play testing. The game is then further optimized to achieve target playability.

Whether the user can adopt the gameplay

Not each and every game is easily understandable. It is a proven and common behavior of users that they do not pay attention to a separate game instruction eplay.

rmal user to
understand the gameplay.

e controls,
gameplay, and game objective. Sometimes, it is mandatory to finish the interactive tutorial to continue playing the game. This is the best possible solution to the problem.

game might
ossible to
predict the time taken by the user to adopt to the gameplay. Thus, it becomes very important to know whether the user understands the game within the planned time or not through play testing. This has a great impact on user retention.

User retention

revenue, which signifies the commercial success of the game. If a user plays the game for the first time, retention has a few segments: daily retention, weekly retention, monthly retention, and so on.

playing
s even
collect data about the time and the specific point in the game where the user left it. This may reveal an issue with the game model. This issue can be rectified to retain more users.

Taking care of the UX

When it comes to the quality of the game, UX or user experience is the most important factor to be considered. Thus, it becomes extremely necessary to polish the UX of the game.

We can categorize UX polishing into the following categories:

- Visual effects
- Sound effects
- Transaction effects
- Action feedback

Visual effects

The user experience of a game is mostly visual. So, each visual effect adds an extra layer of polish. Polishing visual effects means each action feedback should be visual.

not only color
visuals is not enough. This scenario may be improved by introducing visual effects with different shapes of objects or by some other action.

Sound effects

Sound defines the mood of the game. Sound designers design sounds according to the game type. There are two separate types of sound effects:

- Theme music
- SFXs

Theme music

an ambience
for playing the game. Most of the time, it enhances the fun while playing the game
ts the
game.

SFXs

SFXs are the event-based sounds that can be specified for a particular action or event in the game. A few common uses of SFXs are button clicks, user actions, game win, game lose, game start, and so on.

Transaction effects

ne between
nce as the user
has a clear idea about the flow.

ts an in-
of the time,
the user does not pay attention to the numbers and text changes. However, a visible
transaction makes the user notice the numbers.

Action feedback

ystem.
be either
experience.

Android-specific polishing

Android has a specific set of features and limitations. This opens up the possibility for Android-specific polishing. This can be done on the following features or limitations of the Android platform and devices:

- Optimum use of hardware buttons
- Sticking to basic Android features and functionalities
- Longer background running
- Following Google guidelines for Play Store efficiency

Optimum use of hardware buttons

A typical Android mobile or tablet device has the following buttons:

- Home button
- Back button
- Menu button
- Volume up button
- Volume down button
- Lock/Unlock/Power button

is always a
he game.

For example, pressing the Back button should take the user to the previous screen or previous state of the game. The most common use of the Back button for in-game could have

Sticking to basic Android features and functionalities

It is always a good practice to implement basic Android functionalities and use Android-specific features for an Android game. We just spoke about using the device buttons for Android devices.

illing the
e quit.

Longer background running

onventional

way. Rather, Home buttons are used to quickly get out of the game. In that case, the it or the

OS kills the process. The longer it can stay in the background, the quicker the game can be resumed.

Mainly, using low memory and low process overhead can increase the time the game s. In this r experience.

Following Google guidelines for Play Store efficiency

Although Android is an open source platform, Google has some guidelines for Android applications; these are also applicable to games. It is obvious that the Google Play Store is the biggest platform to reach a global audience in the current market scenario. So, it is always a wise decision to follow their guidelines to get featured.

There are several millions of applications available on the Google Play Store. Without getting featured, it is very difficult to attract users to a particular game or application.

Game portability

o its

maximum level without affecting the game itself. In this phase, portability can be increased in three ways:

- Support for various screen sizes
- Support for multiple resolutions
- Support for multiple hardware configurations

Support for various screen sizes

Android has a lot of variety in terms of screen size. The game control system is control

system is also planned according to user convenience.

For Android mobile game development, Android tablet controls are usually a bit different from Android mobile controls. The screen size of tabs is usually bigger should be optimized for both small and big screen scenarios for ease of control.

Support for multiple resolutions

In contrast, there are Android devices that have the same screen size, but different resolutions. In this scenario, the main difference occurs in terms of visibility. So, supporting multiple resolution devices is more art-intensive.

devices.

We have already discussed the variety of resolutions in dpi for Android. So, it is possible to detect the device resolution and use art assets accordingly.

Specification

packages under the same application. So, the developer has the flexibility to more resolutions. Hence, there are several other ways to achieve them.

Integrating a game-specific server is one of the most popular ways to do the job. Developers do not include the major chunk of art assets in the APK. Instead, they put different art packages for different resolutions on a game-specific server. Thus, the game can download specific resolution assets when required. In this way, the developer manages to keep the APK size to a minimum.

Support for multiple hardware configurations

hardware
game on
several configurations smoothly.

Sometimes, the game is optimized specially for some hardware platforms. One of the common examples of such optimization is processor architecture. We have already discussed the variety of processor architectures used in Android games. So, games can be ported for a separate processor architecture.

It is very important to support as many possible hardware configurations as possible to perform such a game polishing function.

Summary

improvisation has no limit. Developers should plan polishing stages and changes to acquire the finished game. This chapter covers each and every section discussed in this chapter.

Last but not least, it should be top-quality so that users pay for it or refer other users. Game quality and features, and keeps users in the game for a longer period.

So far, we have covered almost every aspect of game development for Android. Here are certain parameters to be fulfilled to make a successful game. For these reasons, the game.

We will explore these extra integrations through third parties in depth, and we will try to explore monetization techniques to make the game profitable in the last chapter of this book.

13

Third-Party Integration, Monetization, and Services

Android game development or any other smartphone game development is not
elp the
game spread and perform to reach the next level.

d
n
services to
ur living,
and the gaming industry is not an exception. However, this industry is targeting
entertainment, fun, and interactivity between a device and the user. Developers
elp
e revenue.

Services can be any background support that is not game specific and can improve
ardware and
software programs. Mostly, a server-based service works with the application to
provide the service.

e
following topics:

- Google Play Services
- Multiplayer implementation
- Analytic tools
- Android in-app purchase integration
- Android in-game advertising
- Monetization techniques

- Planning the game revenue
- User-acquisition techniques
- Featuring Android games
- Publishing Android games

Google Play Services

ver, Google

is the owner of the Android OS. So, there can be no one better than Google to be the service provider for the Android platform.

Google Play Services is a background service for all Android devices to access all Google service product APIs. It was launched in 2012 to support Android development and take it to the next level.

The most used services in the Google Play Services package are:

- Google Analytics
- Google IAB
- Google Leaderboard
- Push notifications

Google Analytics

This can

each user's

This analytic

his help, the

developer can improve the game for better experience.

Significance

in the

testing or play testing phase. When the game gets bigger, with a huge user base,

tics helps

in these fields, not only with the current behavior of the user, but with the game performance as well.

Integration tips

vents must
a strategic
ore
data use
and more processing in the game.

The developer should always prioritize events. Events should be tracked based on the game flow design. They should then be validated by user action.

rack when
A simple
revenue.

Best utilization

ny other
o track user
's motive
or intention with the game.

Google IAB

In the modern world of gaming, there are many methods to monetize the
lay Services
comes with the Google In-App Billing tool. This tool is directly associated with
Google Play Store.

side the
the Android
application.

The Google IAB model

We will have
a detailed look at them later in this chapter:

- Consumable items
- Non-consumable items
- Subscriptions

Consumable items

not keep

track of these kinds of items. The most common example of this type of item is in-game virtual currency. Many games are designed around virtual currency, and most of the time, this factor is the backbone of game monetization.

Non-consumable items

track of

ven when

estore the non-

consumable purchases to the user's account.

The most common item under this category is game modes. In many games, there are some open modes and some can be purchased. This system also works with the try-and-buy monetization aspect.

Subscriptions

Subscriptions are basically a time-based model of monetization. This is mainly used in typical service-based applications such as music channels, TV channels, library channels, and so on. Very few games, however, use subscriptions to monetize.

Integrating Google IAB

Google Play Services comes with IAB APIs. The developer needs to register the item IDs,

which are called SKUs. Each SKU represents an item in the Play Store. The developer uses the IAB APIs to integrate the IAB into their game.

Advantages and disadvantages of Google IAB

direct

t and time for

both developers and consumers. Let's have a quick look at the advantages of Google IAB:

- Google IAB provides a direct platform to purchase application components or services within the application
- Google IAB simplifies the monetization aspect of an application
- Google IAB provides multiple options for payment for consumer convenience
- Google IAB stores and manages purchases for non-consumable items

- There is hassle-free implementation and excellent customer support for Google IAB
- The easy refund process is completely managed by Google IAB

ers and consumers or users. However, there are several sectors where Google IAB is still ds to improve:

- Google IAB only provides billing services through Google Play Services
- Google IAB still does not support carrier billing
- Not every user is willing to provide credit card information to Google

Despite these issues, Google IAB is still the most popular platform for billing for Android developers. Google has started including carrier billing services within Google IAB, which may prove to be the most significant feature.

Google Leaderboard

where ard has with an in-built Leaderboard system for Android applications.

Significance

sers compete ology re user d must be chosen carefully.

A good example of a leaderboard-driven game is Candy Crush. Users are very aderboard.

Integrating Google Leaderboard

integrating Google Play Services itself. However, Leaderboard has to be set up in the Google games account to use it.

The developer can choose any parameter or calculation to store leaderboard data.
game.

Most of the developers use this feature efficiently to show different leader lists,

.

Variety of leaderboards

Primary variations of Google Leaderboard are of two types:

- Social Leaderboard
- Public Leaderboard

Social Leaderboard

s circle. For

this feature, the player must log in to their respective Google accounts. This has a

in the same application.

Public Leaderboard

Public Leaderboard stores data for players who choose to post scores publicly.

Otherwise, this data won't be shown by Google Leaderboard, even if they have better score than the existing players on the public Leaderboard.

Options for storing and displaying leaderboards

Leaderboard storage can be classified into two types, based on ascending and descending order. In terms of Google Leaderboard, they are called:

- Larger is better
- Smaller is better

A score is always a numeric value, which is again classified into three formats:

- Numeric value format
- Time format
- Currency format

In the case of a numeric value, the developer can specify the decimal placement. In the case of the time format, the developer needs to pass the score in milliseconds, and it will be automatically interpreted in the *hh:mm:ss* format. In the case of the currency

eforehand.
ecified unit
format.

Leaderboard can have unique icons to display or indicate a unique leaderboard.

Push notifications

The push notification service can be achieved through the **Google Cloud Messaging (GCM)**

There are primarily four components used to implement push notifications for Android using GCM:

- Database
- Server
- Target device
- GCM service

Database

e with the GCM
service. So, each device is required to register only once. The same details are used to send push notifications to the registered target devices.

Server

Developers need to put up a server to achieve and control push notifications.

Target device

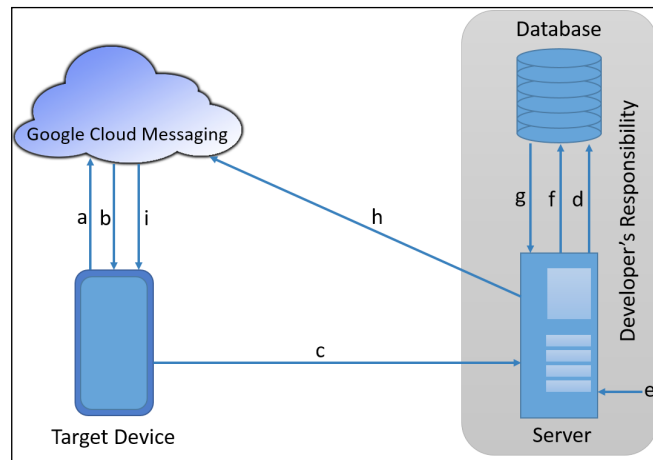
om the
thout
registration, a target device cannot receive any notifications.

GCM service

ges

IDs and

customized messages. GCM is only responsible for pushing the given content to specified devices:



Workflow of the push notification system using GCM

Now, let's discuss the push notification workflow. In the preceding diagram, the push notification system works according to the indicated indexes (for example, **a**, **b**, **c**, and so on):

1. **a**ion
ID and sender ID.
2. **b**: GCM sends the registration ID back to the sender after a successful registration.
3. **c**: The device sends the registration ID to the developer's server.
4. **d**: The server stores the registration ID to the database.
5. **e**: The developer initiates the process to the push notification with customized content.
6. **f**: The server fetches the registration ID list from the database.
7. **g**: The database provides all the registration IDs.
8. **h**: The server requests GCM with developer-specified content and registration IDs.
9. **i** to
their registration IDs.

Integrating push notifications

Integrating push notifications is done in three steps:

1. Application integration
2. GCM setup
3. Server setup

Application integration

the medium

GCM

communication services.

It requires a set of manifest permissions:

```
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.GET_ACCOUNTS" />
<uses-permission android:name="android.permission.WAKE_LOCK" />
<uses-permission android:name="com.google.android.c2dm.permission.
RECEIVE" />
<permission android:name="com.example.gcm.permission.C2D_MESSAGE"
    android:protectionLevel="signature" />
<uses-permission android:name="com.example.gcm.permission.C2D_MESSAGE"
/>
```

The manifest will also require declaration of the GCM receiver and GCM service:

```
<receiver android:name="com.google.android.gms.gcm.GcmReceiver"
    android:exported="true" android:permission="com.google.
android.c2dm.permission.SEND" >
<intent-filter>
<action android:name="com.google.android.c2dm.intent.REGISTRAION" />
<action android:name="com.google.android.c2dm.intent.RECEIVE" />
<category android:name="com.example.gcm" />
    </intent-filter>
</receiver>
<service android:name=".GcmService" android:exported="false">
    <intent-filter>
<action android:name="com.google.android.c2dm.intent.RECEIVE" />
    </intent-filter>
</service>
```

M.

e. We

will follow the simplest processes within the main Android activity and store the registration ID for one-time registration of the application.

Here are the required declarations:

```
private final Context testContext = this;
private final String SENDER_ID = "<Application ID from Google
developer console>";
private final String SHARED_PREF = "com.test.gcmclient_preferences";
private final String GCM_TOKEN = "testgcmtoken";
```

The registration code should be put inside `onCreate()`:

```
SharedPreferences appPrefs = testContext.getSharedPreferences(SHARED_
PREF, Context.MODE_PRIVATE);
String token = appPrefs.getString(GCM_TOKEN, "");
if (token.isEmpty())
{
    try
    {
        InstanceID instanceID = InstanceID.getInstance(testContext);
        token = instanceID.getToken(SENDER_ID,
            GoogleCloudMessaging.INSTANCE_ID_SCOPE, null);
        if (token != null && !token.isEmpty())
        {
            SharedPreferences.Editor prefsEditor = appPrefs.edit();
            prefsEditor.putString(GCM_TOKEN, token);
            prefsEditor.apply();
        }
    }
    catch (Exception e)
    {
        e.printStackTrace();
    }
}
```

Now, let's define `GCMService.java` to handle the GCM message:

```
public class GcmService extends GcmListenerService
{
    @Override
    public void onMessageReceived(String from, Bundle data)
    {
        JSONObject jsonObject = new JSONObject();
```

```
        Set<String> keys = data.keySet();
        for (String key : keys)
        {
            try
            {
                jsonObject.put(key, data.get(key));
            }
            catch (JSONException e)
            {
                e.printStackTrace();
            }
        }
        try
        {
            sendNotification("Received: " + jsonObject.toString(5));
        }
        catch (JSONException e)
        {
            e.printStackTrace();
        }
    }

    @Override
    public void onDeletedMessages()
    {
        Log.d("Message is deleted ...");
    }

    @Override
    public void onMessageSent(String msgId)
    {
        Log.d("Message is sent ..." + msgId);
    }

    @Override
    public void onSendError(String msgId, String error)
    {
        Log.d("Sending Error ... Msg" + msgId);
        Log.d("Error ..." + error);
    }
    private void sendNotification(final String msg)
    {
        Log.d("Sending Msg ..." + msg);
    }
}
```

GCM setup

the
cation ID,
which is required to configure the push notification system.

Here are the steps to enable GCM for the Android project:

1. Create a project on the Google Cloud platform.
2. Use the Google API to generate an API key.
3. Create a server key for Android.
4. Enable GCM for the application.

Server setup

Notification server development can be implemented by any cloud connection server
sifying the
following criteria:

- The application server should be able to communicate with the application
- The application server should be able to send properly formatted requests to the GCM connection server
- The application server should be able to handle application requests and resend them using exponential backoff
- The application server should be able to store the API key and client registration tokens in a secured database

Significance of push notifications

Push notifications are an inevitable part of modern day game development. Push notifications are used for the following reasons:

- User retention
- User control
- Knowing user behavior
- Alternative communication channel

User retention

Push notifications provide users with current updates and information on the game. There are many cases where the user downloads a game and then forgets about it. Sometimes, users leave games in between. Push notifications help these users regain interest in the game. This procedure improves user retention.

User control

Through device settings and the notifications center, the developer can control the y.

Knowing user behavior

Using user controls, the developer can track user behavior upon notifications. likes and dislikes.

Alternative communication channel

There are several ways to communicate with end users. Mostly, users do not often communicate with the developer. So, a one-way communication channel proves to be fruitful. A push notification system fits the role perfectly. It is the best possible medium to deliver messages about the latest news, updates of the game, offers, and features. In some design models, it can be used to deliver game status information to users.

Multiplayer implementation

sole

gaming. The modern day gaming industry consists of extensive use of social networking. This automatically opens up the opportunity for multiplayer gaming.

Improved hardware systems and continuous network support with modern gaming

can be classified mainly into two categories:

- Real-time multiplayer
- Turn-based multiplayer

Real-time multiplayer

Real-time multiplayer is just like playing sports together, where every player reacts to any action by the game or other players at the same time. For example, a football situation at

the same time. If we imagine the same scenario from a digital gaming perspective, it will be called a real-time multiplayer.

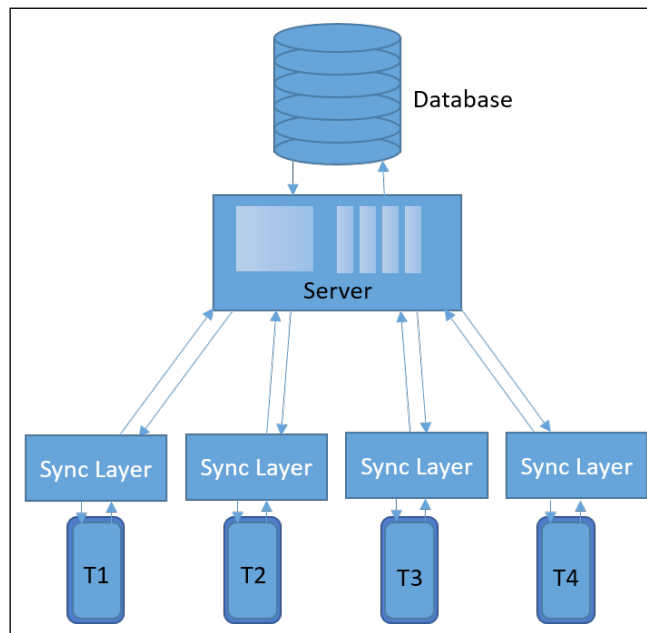
where the

with the

database when required, and the user does not have any control.

The client or terminal devices are the only medium that interacts with users. However, in many cases there are some extra layers used on the client side to perform a few actions without server validation to keep the spontaneity of the real-time multiplayer game.

Let's look at the general architecture of the real-time multiplayer system:

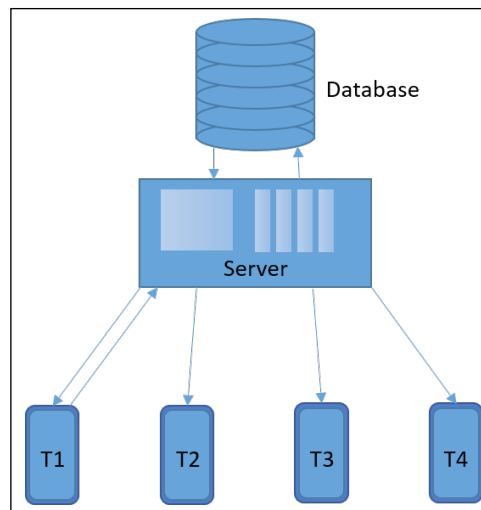


sync layer
the terminal
device application and server.

Turn-based multiplayer

to play at
n, the other
player remains idle.

r. A server
shown in
f the
ticular terminal
devices and should only listen to that device. Let's have a look at the architecture
diagram:



There are more types of multiplayer models possible in Android gaming. Until now, we have discussed only the models implemented over the Internet. Local multiplayer gaming is also possible in Android. We can classify these games into the following categories:

- Single-screen real-time multiplayer
- Pass and play turn-based multiplayer
- Local network multiplayer

Single-screen real-time multiplayer

For single-screen real-time multiplayer, it is recommended that the developers target large-screen devices to provide more space for control for multiple players at a time.

Pass and play turn-based multiplayer

In this model, all players take turns to play the game. The game state is synchronized with the game turn.

Here, one player passes the same device to the next player after playing their turn. Then, the next player reacts to the current state of the game. In this model, the game state does not change until the completion of each turn.

uired, and
ce memory.

Local network multiplayer

g local
t act as a
uetooth,
Wi-Fi, or an infrared connection.

Analytic tools

We have already discussed Google Analytics. There are several other analytics
nt of view.

d users
onetization.

Requirement of analytics tools

rms
many
products. This data helps the developer or manufacturer modify or improve the
product.

through the
following points:

- User behavior
- Game crash reports
- Game event triggers
- Gameplay session timing
- Gameplay frequency
- Game balancing
- User retention
- Piracy prevention

User behavior

Analytic tools can track each and every movement of every user. This data can be havior validates the **meta** design of the application or game.

Game crash reports

h cause and location. However, an encoded package of game code cannot reveal the location completely, where the class and its members are encoded in meaningless symbols. he crash.

Game event triggers

The developer can set triggers from the game itself to track any or every aspect of the game. These can be any event inside the game. It is a common practice for game developers to use this trigger system for the game start, game end, and a few strategic events such as IAP, advertisement display, mode selection (if any), and so on.

Gameplay session timing

Analytic tools track gameplay timing by triggering two events between the application going to the background or the application exit. By calculating the time in between, er was inside the application in a single session.

Gameplay frequency

So, developers can have the data increase or improve the sessions. Developers can frequency, or monthly frequency.

Game balancing

Developers can collect data on user scores and playtime to detect the difficulty for every single player has a different ability and skill to play. Thus, developers must set some standards to balance the game properly globally.

User retention

User retention is one of the most important aspects for developers to generate revenue from the game. This means the number of users playing the games repeatedly. User retention can be also time based, such as daily, weekly, and monthly.

Piracy prevention

In the case of Android gaming, there might be a model of premium or freemium games. In this model, the user buys the game or some components inside the game rs. They can e paid game or paid components for free. Piracy is a major problem for developers in terms of generating revenue.

ate the purchase, which adds an extra security layer to prevent game piracy.

Monetization aspects of analytic tools

eatures are
developers
revenue
nalytic data.

actions:

- Identify popular regions of the game
- Identify a user's likes and dislikes
- Validate and improve the metagame
- Track paying users
- Track and count advertisement display

Identify popular regions of the game

more
lly for free
or freemium games, it is extremely necessary to find the part of the games that users are visiting frequently.

Identify a user's likes and dislikes

and
ay test
on a decent amount of users, it is very hard to predict a user's likes and dislikes.

that users
like or dislike. Developers can change the strategy or plan for a better update for the game.

Validate and improve the metagame

ame.
Only
analytic tools can validate this prediction after launching the game.

Track paying users

lly paying
me revenue.

Track and count advertisement display

vertisement
display. Thus, it becomes easier to predict revenue from advertisements and the developer can even plan for better filling of advertisements.

Some useful analytic tools

e are
many analytic tools available in the market that are as good as Google Analytics and can be a good option for replacement. There is no restriction for developers in terms d the developer
can even integrate multiple tools for different purposes.

Let's have a quick look at such tools:

- **Flurry** (<https://dev.flurry.com>)
- **GameAnalytics** (<http://www.gameanalytics.com/>)
- **Crashlytics** (<https://fabric.io/kits/android/crashlytics>)
- **AppsFlyer** (<https://www.appsflyer.com/>)
- **Apsalar** (<http://support.apsalar.com/>)

- **Mixpanel** (<https://mixpanel.com/android-analytics/>)
- **Localytics** (<https://docs.localytics.com/index.html>)
- **Appcelerator** (<http://www.appcelerator.com/mobile-app-development-products/>)

Flurry

each and
stall, and the
developer can start getting data right away.

GameAnalytics

It helps
you understand player behavior and build better games through analytics data on a
dynamic dashboard typically designed for games.

Crashlytics

Crashlytics is the most powerful and efficient bug-reporting tool. It can intercept any
lightweight
and easy to use for developers.

AppsFlyer

ol with
analytics features. It basically uses AppsFlyer's **NativeTrack™** to provide analytic
support for games.

Apsalar

Apsalar is mostly used for advertising attribution. It gives a good look at the game
marketing ROI. It also helps find out which marketing campaigns are working
and which ones need to be avoided. They also offer great marketing tools such as
SmartTagss.

Mixpanel

Mixpanel's benefit is mainly for non-technical people who can easily create custom
segments
users and see which segments are working best for the game.

Localytics

Localytics provides most of the functions for data analysis. The platform provides real-time analytics, remarketing data, attribution, and more. Localytics's messaging features differ from other general analytic tools.

Appcelerator

and analytics.

, which can be

used on multiple platforms and provide immediate insight into the five key mobile metrics: retention, engagement, adoption, quality, and conversion.

Android in-app purchase integration

an be

bought from inside the application with the help of several payment gateways.

This is one major aspect of monetization for Android games.

What are in-app purchases?

In the modern day gaming industry, freemium games are booming. This means users can play the game for free, but they have to pay for certain components or for game progression advantages. This model has been proved to be a success, as it supports both free and premium concept in terms of digital gaming.

In-app purchases serve this purpose perfectly. We have already discussed Google In-App Billing services, which is just a means of in-app purchasing through Google. But there are other services that support the same thing.

oice to buy

the following types of content:

- Unlock certain features in the game
- Buy certain items to get an advantage over other players
- Unlock some modes inside the game
- Increase ease of play
- Remove annoying advertisements

ame game
ll of them
e some
money.

In-app purchase options

rchases.
rent service
pay, but if
f purchasing.

It is always a good practice to provide the maximum possible options to the user for s:

- All users might not have a credit card
- All users might not have a debit card
- All users might not have activated net banking
- All users might not have sufficient talk-time balance
- All users might not like to directly use real currency

The developer should provide the maximum possible options to overcome these issues and make users use real cash for the game. Currently, available billing services o two major divisions:

- Store billing services
- Career billing services

Store billing services

oads the
game. The game should be connected to a store with provided APIs in order to access this feature. We already discussed that Google IAB is a type of store billing service that includes several methods of paying, including credit card, debit card, selective career billing, and so on.

most
mentionable store billing, other than Google, is Amazon billing service, which provides almost the same features as Google.

Amazon billing services

Amazon billing service works exactly like Google IAB. However, API and integration is slightly different to Google IAB.

The developer needs to include the `com.amazon.device.iap` package to integrate Amazon IAP. This process has mainly three components:

- `ResponseReceiver`
- `PurchasingService`
- `PurchasingListener`

ResponseReceiver

Amazon IAP is an asynchronous process. It works as a background service that requires a response receiver to be implemented. The developer needs to declare the receiver in the manifest file.

PurchasingService

The `PurchasingService` class is used to retrieve various types of information about out the fulfillment of a purchase.

PurchasingListener

The `PurchasingListener` interface is used to process asynchronous callbacks from which is why d.

Amazon IAP is similar to Google from a feature and integration point of view. There nother option pers of Android games prefer to stick to mainstream billing services.

Career billing services

r billing means the user pays developers for in-app products from their mobile balance, which is managed by the connection provider.

Currently, Google IAB has started supporting career billing within store billing.

Types of in-app purchases

pes of products depend on game design and game genre. The types are:

- Consumable items
- Non-consumable items
- Subscription

Consumable items

of Google IAB, these products are termed non-managed products.

the user.
genres of this type of products. Users can buy the same item multiple times.

Consumable items must be defined on the billing server to make them understandable to the billing service.

Non-consumable items

s keep track of these purchases.

then upon
eds to buy
this product only one time throughout the application's life.

Subscriptions

ication. There are very limited uses for subscription in games. However, this is a good option to provide some feature or services for a limited time or limited use.

ever there is a renewable feature that allows the user to subscribe again for the same thing upon expiration of the service period.

Android in-game advertisements

In-game advertisements are the most significant factor in monetization for both free and freemium games. Developers use their game platform to show advertisements in order to generate revenue.

Here is how it works:

1. Advertisers submit the advertisements to various advertisement agencies.
2. Each advertisement has a certain value and time duration limitation, which is called campaign cost and campaign time, respectively.
3. The developer subscribes with those agencies.
4. The developer integrates the agency advertisement platform to include and show advertisements.
5. The developer sets the parameters for advertisement types, genre, and level.
6. When the application triggers an advertisement call to the agency server, the criteria predefined by the developer.
7. Upon successful match, the server sends the advertisement elements to the client device application.
8. The application loads the advertisements.
9. The application shows the advertisements on request.
10. The server keeps a count of successful display of advertisements and calculates revenue as per the campaign cost.
11. The developer receives the revenue after meeting certain criteria from agencies.

Requirement for advertisements

s or sponsorships. We will only look at advertisements here. Let's understand the requirement for advertisements inside a game.

We all work to earn our living. Android is an open source platform, and most of its user base consists of free users. This means developers have only one option left. rm to depend on.

Advertising as an industry is old and has proven its sustainability in the market. In-it is always a win-win situation for both developers and advertisers.

Terminologies in advertisement monetization

Now, we will discuss typical game advertisement platforms. The developer needs to be familiar with a few terms to get used to in-game advertisement:

- eCPM
- CPC/CPA
- CPI
- RPM
- Fillrate

eCPM

eCPM stands for **effective cost per mile**, which is the result of a calculation of number of ad impressions of that banner or campaign expressed in units of 1,000, which is represented by the letter *M* at the end.

CPC/CPA

CPC stands for **cost per click**, which means the developer will earn a certain amount if the user clicks on displayed advertisements. **CPA** stands for **cost per action** which is similar to CPC.

CPI

CPI stands for **cost per impression**, which means the developer will earn a certain amount if any advertisement is successfully displayed inside an application. Generally, these earnings are lower than CPC.

RPM

RPM stands for **Revenue Per Mile**. It indicates the total revenue generated from a thousand interstitial advertisements. RPM includes all types of revenue models. RPM is calculated by the following formula:

$$RPM = (Total\ revenue) / (Ads\ served / 1000)$$

Fillrate

Fillrate is the percentage of successfully served advertisements by the server. We already know that the application requests the advertisement server for advertisements. This is termed a "request." If a server successfully serves advertisements upon request, then the advertisement is termed an "impression". So we have our fillrate, as follows:

$$\text{Fillrate} = (\text{Impressions/Requests}) * 100\%$$

Types of advertisements

id games:

- Banner advertisements
- Interstitial advertisements
- Video advertisements
- In-game dynamic advertisements

Banner advertisements

rs cannot

CPC is

acceptable. Many developers nowadays avoid using banner ads, as it occupies a significant space of the game screen. Banner ads are displayed at a given rectangular shape at the edge of visible display.

Possible banner display positions are as follows:

- Top left
- Top center
- Top right
- Bottom left
- Bottom center
- Bottom right

ng table:

Banner type	Target	Size in pixels
Standard banner	Phones and tablets	320 x 50
Large banner	Phones and tablets	320 x 100
IAB full-size banner	Tablets	468 x 60
IAB leaderboards	Tablets	728 x 90
Smart banner	Phones and tablets	Screen width x 32 Screen width x 50 Screen width x 90

Interstitial advertisements

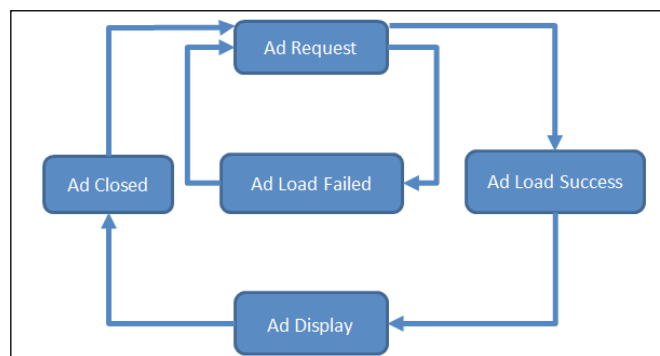
ment based on various campaigns. Normally, an interstitial has a defined close button for users to close the advertisement and go back to the game.

ushing
gers an
interrupt for the game thread.

This type of advertisement is widely being used in games because of decent revenue. Game monetization design has a significant role in interstitial advertisements. Each advertisement placement has to be strategically based on the analytic data.

Integration best practice

Integrating interstitial advertisements should follow a few logical ad displaying cycles:



It is always a good practice to follow the cycle. An ad should be loaded and be in
ext ad
should be loaded immediately to avoid load delay.

Video advertisements

e. This
video
are two types
of ads:

- Full length ads
- Short length ads

Full length ads

Full length ads are generally longer. These types of ads are generally skippable,
which provides an option to skip after a certain amount of time.

Short length ads

Short length ads are comparatively smaller and have no option to skip.

In-game dynamic advertisements

This concept provides an option to show any available ad banner within the
predefiism
resizes the ad in the given size within the application.

Monetization techniques

any
application. The developer needs to decide their game monetization model based on
:

- Premium model
- Free model
- Freemium model
- Try-and-buy model

Premium model

ame before

-game

advertisements. This is just a one-time buy for the user for the gameplay, and normally all users have the same opportunity for game progression.

Free model

rtisements

es not have

any extra privileges for any actions.

Freemium model

ly without

es to

provide extra content or facility for game progression.

Try-and-buy model

model.

res.

The free version usually has limited content or limited use. This version may or may not be a typical premium game model. Sometimes, developers use in-app purchases within the free version to do the job, as needed.

Planning game revenue

ion.

nue

or having strong financial support. Let's discuss game planning now, to keep a developer developing games.

Revenue versus profit

Most new game developers do not know that revenue and profit are two different things.

users.

may

After all the

required payments and cuts, the remaining amount is called profit. So, high revenue does not mean high profit.

However, without generating revenue, there cannot be any profit. So, the developer must plan revenue in order to generate profit.

Revenue sources

Now, we know that generating revenue is necessary. To generate revenue, the developer must know about the possible revenue sources. We will discuss the main sources here:

- Advertisement revenue
- In-app purchase revenue
- Other sources

Advertisement revenue

Especially for free and freemium games, advertisements are one of the main sources of revenue. Advertisements are often placed through various marketing campaigns.

There is another platform called ad mediation. This platform provides advertisements from different agencies. Sometimes, this platform helps find the highest rate among available advertisements. This special feature is called real-time bidding.

In-app purchase revenue

This is a way of generating revenue for mainly freemium game models. The revenue is kept by the developer. After keeping a portion of the revenue, the developer spends extra bucks to get a strong hold of the game, they use in-app purchases.

design and market behavior. Some game models demand content, some demand features, and a few demand both.

s, which we have discussed. However, choosing a particular service may have an effect on before tying the knot with a billing service provider.

Other revenue sources

of revenue too. Offerwall and coupon systems are two other options. The developer might opt for more money. Point of view.

As the industry modernizes, new sources of revenue may come up to help developers grow and make better games.

Regional variations of revenue plan

s with e, then they must consider these factors in a revenue plan.

me or in a single he basis of region. It has been established in the market that user behavior varies a lot based on region.

For example, Asian user action and behavior may vary from African or American users, so does the spending capacity. So, the developer should plan game revenue some

regions, users do not pay real cash. In those cases, the developer must have a different approach to generate revenue.

User base variations

rating is one
rating
ney, and
inside a racing
game helps users save some time for game progression, it might not work in all
instead of
paying. The developer has to have a plan to convert that play time into revenue by
some means.

User behavior variations

. One of the
ous and hot
ted to the
this game
ar among
of the
game.

User acquisition techniques

If a game has no users, it is as good as scrap. This does not mean that the game
ion
ich is
increasing the competition.

In this immense crowd, a single game may disappear, irrespective of its quality, but
sful if it
has a significant number of users and good retention.

following
topics:

- Game promotion channels
- Game blogs and forum discussions
- Paid user acquisitions
- Other techniques

Game promotion channels

game

promotion channels that advertise in various mediums. A specific game genre channel promotes the same kind of games. Let's look at few of these mediums:

- YouTube channels
- Android forums
- Sports forums
- Facebook promotions
- Twitter and other social platforms

YouTube channels

.

each these

o know

about the game.

A good game review from such channels can get developers a significant amount of users. However, such channels might charge developers for reviewing their games. Thousands of users may be found from such channels.

Android forums

There are hundreds of Android forums available, and there are thousands of active participants that can be found talking about games, apps, development style and standards, and so on. Such forums are also good platforms to promote Android games. However, developers should be specific to the topic, and the game should have the potential to be talked about. A few hundreds of users can be achieved through such channels.

If a developer uses any Android-specific special feature and has implemented reach
ations.

Sports forums

There are many forums for specific sports. This method works mostly for games
um about
games of the same sport. For example, if a developer has made a cricket game, then the game should be posted and promoted through cricket forums.

This method has an added advantage. As the forum is specific to the same sport, then the developer might find a few people who are experts in the sport and can share their valuable opinion about the game, which may make the game better.

Facebook promotion

s. It is a common practice for developers to use this platform for promotion of games. Social networking can find a significant number of users for a game.

eveloper.

This page is one of the communication mediums between the user and the developer. me so that new users have a good idea about the game even before they start playing it.

Twitter and other social platforms

Twitter and other social platforms are also useful for game promotion and increasing crease the user count.

A social platform need not necessarily be a digital or web platform. It can be nts to showcase their games or participate in various events and competitions to get recognition. Good recognition for a good game can help gain more users.

Game blogs and forum discussions

isition.

so that

there can be a significant number of people who will participate in the discussion.

their

opinions, criticism, or suggestions for the game. This can make a game famous, which always helps gain users.

Paid user acquisition

There are many marketing agencies that find users for games. Usually, such agencies h to spend real cash to gain users, then this is probably the best possible solution.

search

repay

the developer with more revenue. Sometimes, the wrong choice of promotion and a wrongly acquired user base may lead the game to disaster.

Other techniques

s.

of

them are as follows:

- Many times, the developer approaches users individually to promote games
- Many times, the game is promoted verbally through friends and family
- Many times, developers run campaigns for the game
- The developer may approach a good publisher to get help acquiring more users
- Sometimes, celebrities are used to promote the game

There is no fixed path for promoting a game and acquiring users. It is always a good habit to keep all the options open and aim for the maximum possible outcome.

User retention techniques

Creating a good user base might not be enough to generate decent revenue to gain profit out of the game. Hence comes the term **user retention**. This means the number of users who are playing the game repeatedly.

Users may download a game, and after a few game sessions, they may never come game.

User retention is calculated on time parameters such as weekly or monthly use. This means how many users are coming back to the game within the given time period.

the

everal

for doing

business with games.

metagame.

Let's discuss the major techniques through these points:

- Daily bonus
- Leaderboards and achievements
- Offerwall integration
- Push notifications
- Frequent updates

Daily bonus

Back to the
elements
and elements.

This system motivates users to keep coming back to the game. Thus, a developer gets more game session time to convert it to revenue.

Leaderboards and achievements

Give users
in the
game, users must come back to the game and spend time within the game.

Offerwall Integration

The developer uses some real-world offers to keep users inside the game. Real offers such as coupons and discounts always interest users. It provokes them to come to the offerwall frequently. Offerwalls not only help retain users, but also help generate more revenue from various offer campaigns.

Push notifications

Push notifications can inform users about the latest information and updates about the game. Even if the user is not playing the game, push notifications help them gain interest in the game, which may make the user start playing again.

Sometimes, a user downloads a game and forgets about it. In such a case, a push notification
progress inside the game.

Frequent updates

The developer should keep updating the game frequently to keep up with the chart and helps retain existing users.

ion for a

.

Featuring Android games

A successful game means both profits and fame. A game can be profitable with a hat easy.

A game becomes famous if it gets featured in various places.

eria:

- Creativity and uniqueness
- User reviews and ratings
- Download count
- Revenue amount

Creativity and uniqueness

There
talk about
ediums.

gn, and

playability. A good art style, good design, and playability can make a game get
r can make

a game famous, which may lead to more users and revenue.

User reviews and ratings

After publishing the game, the game's fate depends on users. New users cannot be
. Hence,

ways

ctively

ratings and

e early stages

hey perform

well in the later stages.

Download count

Download count is another game featuring criteria. As soon as the download count increases, there is more probability that the game will get featured by the store. The developer should concentrate on increasing the number of downloads as soon as possible to get featured or to be in the top list.

Revenue amount

An Android game can be featured with the amount of revenue generation in the top the game or the game is generating a significant amount of revenue. Getting featured in the top tly generates more download count and revenue. However, to remain in the top list, the developer on user retention.

Publishing Android games

the game.
However, these are the steps after the game gets published in the market. There are two ways through which the developer can publish the game:

- Self publishing
- Publishing through publishers

Let's have a quick look at this segment of game development.

Self publishing

blishing.
e game IPs
Developers
take full responsibility for the game, game ratings and reviews, game revenue, and user satisfaction.

Publishing through publishers

ity after
ishers to
tions, and
keep up
with the conditions to enjoy less responsibility and better marketing.

Summary

lopers
should be capable of taking the right decision for the game to taste success. It is a well-known fact that success does not come easily. This chapter shows all the factors of a game that need to be taken care of to achieve success.

king
ame's
t capable of
doing it on their own. Using social platforms is also a must.

Finally, choosing the right publishing place and targeting the correct audience for the game can bring success. In the case of Android-specific gaming, there are already
viders
he game has
been made. Otherwise, there is a strong possibility that a good game might be lost in the crowd of millions of Android games.

Index

Symbols

2D/3D performance comparison

- 3D processing, heavier than 2D processing 151

- about 151

- different look and feel 151

2D assets optimization

- about 147

- data optimization 147

- process optimization 148

- size optimization 147

2D rendering pipeline 145

3D assets optimization

- about 148

- model optimization 148

- polygon count, limiting 148

3D rendering pipeline 146

A

Acorn RISC Machine (ARM) 270

action feedback, UX polishing 281

ADT-1 27

advantages, C++ for games

- CPU architecture support 262

- cross-platform portability 261

- faster execution 262

- universal game programming language 261

advertisement monetization, terminologies

- CPC/CPA 312

- CPI 312

- eCPM 312

- fillrate 313

advertisements styles, for Android games

- about 313

- banner advertisements 313, 314

- in-game dynamic advertisements 315

- interstitial advertisements 314

- video advertisements 315

Ahead-of-time (AOT) compilation 2

allocation tracker 220

Amazon billing services

- about 309

- PurchasingListener 309

- PurchasingService 309

- ResponseReceiver 309

analytic tools

- about 302, 305

- Appcelerator 307

- AppsFlyer 306

- Apsalar 306

- Crashlytics 306

- Flurry 306

- GameAnalytics 306

- Localytics 307

- Mixpanel 306

- monetization aspects 304

- requisites 302

Android

- about 2

- future, in VR 238

Android application

- life cycle 6, 7

- memory management 7

- performance 7

Android build process

- native shared library 256

- native static library 257

Android consoles

- development insights on 42
- exploring 28-33
- GamePop 32
- Game Stick 31
- Mad Catz MOJO 32

Android DDMS

- about 211
- connecting, to Android device
 - filesystem 212
- device operations, emulating 214, 215
- heap information monitoring 213
- log information, tracking with Logcat 214
- memory allocation, tracking 213
- network traffic, managing 214
- network traffic, monitoring 214
- profiling methods 213
- thread information monitoring 213

Android Debug Bridge (adb)

- about 53, 250
- client, on development machine 53
- daemon 53
- server, on development machine 53
- using, on Android device 54

Android Development Tool (ADT) 50

Android device

- Android Debug Bridge (adb), using on 54

Android device debugging

- about 217
- usage, of breakpoints 217

Android device filesystem

- Android DDMS, connecting to 212

Android device testing

- about 215, 216
- full/complete testing 216
- prototype testing 216
- regression testing 216
- release testing/run testing 216

Android-enabled STB devices

- Arcadyan BouygtelTV 25
- Forge TV 25
- Freebox Mini 4K 25
- LG UPlus Android TV 25
- OgleBox Android TV 25
- Shield Android TV 25

Android game development

- about 1, 2
- challenges 4
- design constraints 5
- features and support 3
- user experience 4

Android games

- life cycle 6, 7, 101, 102
- memory management 7
- performance 7
- publishing 325
- publishing, through publishers 326
- self publishing 325
- success, reasons 2

Android games, featuring qualities

- about 324
- creativity and uniqueness 324
- download count 325
- revenue amount 325
- user reviews and ratings 324

Android in-app purchase integration 307

Android in-game advertisements

- about 311
- requisites 311
- working 311

Android library shaders

- about 160
- BitmapShader 160
- ComposeShader 160
- LinearGradient 160
- RadialGradient 160
- SweepGradient 160

Android.mk file, options

- BUILD_SHARED_LIBRARY 259
- BUILD_STATIC_LIBRARY 259
- CLEAR_VARS 258
- LOCAL_MODULE 259
- LOCAL_MODULE_FILENAME 259
- LOCAL_PATH 259
- LOCAL_SRC_FILES 259
- PREBUILT_SHARED_LIBRARY 259
- PREBUILT_STATIC_LIBRARY 259
- TARGET_ARCH 259
- TARGET_ARCH_ABI 259
- TARGET_PLATFORM 259

- Android mobiles**
 - development insights on 36, 37
 - exploring 18-20
- Android NDK**
 - about 256
 - working 256
- Android PackageManager 14**
- Android programming structure**
 - about 76
 - call hierarchy 77, 78
 - class formation 76
- Android RunTime (ART) 2**
- Android SDK**
 - about 50, 257
 - used, for creating sample game loop 97-101
- Android-specific polishing**
 - about 282
 - Android functionalities and features, implementing 282
 - Google guidelines, following for Play Store 283
 - longer background running possibility 283
 - optimum use, of hardware buttons 282
- Android STBs**
 - development insights on 39, 40
 - exploring 24-28
- Android Studio**
 - about 65
 - Android project view 65, 66
 - memory and CPU monitor view 66
- Android tablets**
 - development insights on 38, 39
 - exploring 22-24
- Android televisions**
 - exploring 24-28
- Android TV game development**
 - development insights on 39, 40
- Android Virtual Device (AVD)**
 - about 51, 207-209
 - attribute factors 209
 - configuring 51-53
 - dedicated disk space 52
 - hardware profile 51
 - other features 52
 - system image mapping 51
- Android VR development best practices**
 - about 248
 - better audio experience 250
 - draw call limitations 248
 - overheating problems, overcoming 249
 - proper project settings, setting up 250
 - proper test environment, using 250
 - steady FPS, keeping 249
 - triangle count limitations 249
- Android VR game market, challenges**
 - about 250
 - limited device support 251
 - limited game genres 251
 - long game sessions 251
 - low target audience 251
 - real-time constraints 252
- Android VR games**
 - about 237
 - current industry situation 238
 - history 237
 - technical specifications 237, 238
- Android watches**
 - development insights on 42
 - exploring 33-35
- animation polishing 275**
- Appcelerator**
 - about 307
 - reference 306
- application memory distribution 188**
- Application.mk file, options**
 - APP_ABI 260
 - APP_BUILD_SCRIPT 260
 - APP_CFLAGS 260
 - APP_CPPFLAGS 260
 - APP_LDFLAGS 260
 - APP_OPTIM 260
 - APP_PLATFORM 260
 - APP_PROJECT_PATH 260
 - APP_STL 260
 - NDK_TOOLCHAIN_VERSION 260
- application priority**
 - about 188
 - active process 189
 - active services 190
 - background process 190
 - visible process 190

- applications**
 - versus games 5
- applications, as game**
 - qualifying criteria 6
- application services**
 - about 191
 - life cycle 191
 - misconceptions 191
- AppsFlyer**
 - about 306
 - reference 305
- Apsalar**
 - about 306
 - reference 305
- Arcadyan BouygteITV 25**
- art assets 203**
- ART message log 218**
- art optimization 176**
- art polishing**
 - about 275
 - animation polishing 275
 - marketing graphics 275
 - UI polishing 275
- asset optimization tools**
 - about 60
 - full asset optimization 60
 - sprites, creating 61
- ATC 267**
- attribute factors, Android Virtual Device (AVD)**
 - Android target version 210
 - Android version API level 210
 - AVD display size 210
 - AVD resolution 209
 - CPU architecture 210
 - extended AVD settings 211
 - hardware input options 211
 - name of AVD 209
 - other options 211
 - RAM amount 210
- audio assets 203**
- Avatar 236**
- B**
- banner advertisements 313, 314**
- base port 56**

- best optimization practices**
 - about 201
 - asset-using techniques 202
 - cache data, handling 204
 - data structure model 202
 - design constraints 201
 - development optimization 201
- best practices, for making Android game**
 - about 10
 - background behavior 13
 - battery usage, maintaining 14
 - extended support, for multiple visual quality 15
 - game quality, maintaining 11
 - interruption handling 13
 - maximum devices, supporting 12
 - maximum resolutions, supporting 12
 - minimalistic user interface 11
 - multiplayer, introducing 15
 - social networking, introducing 15
- best testing practices**
 - about 230
 - APIs 230
 - testing techniques 231
 - tools 230
- BitmapShader 160**
- build dependency**
 - about 257
 - Android SDK 257
 - C++ compiler 257
 - Cygwin 258
 - Gradle 258
 - Java 258
 - Python 258
- C**
- cameras**
 - first-person camera 145
 - fixed camera 145
 - moving camera 145
 - rotating camera 145
 - third-person camera 145
- Cardboard**
 - headset components 241
- Cardboard application**
 - upgrades 241

- variations 241
- working principle 241
- Cardboard development styles**
 - display properties 243
 - game controls 244
 - in-game components 243
 - VR device adaptation 243
 - VR game, exiting 242
 - VR game, launching 242
- Cardboard SDK**
 - about 240
 - basic guide, for developing games 242
- career billing services** 309
- C++ compiler** 257
- C++, for games**
 - about 261
 - advantages 261
 - conclusion 263
 - disadvantages 262
- Coby Kyros MID7047**
 - configuration specifications 23
- Cocos2d-x**
 - about 68
 - cons 69
 - pros 68
- color resources** 192
- common game development mistakes**
 - about 149
 - shortcut, during development 150
 - substandard programming, using 150
 - use of full utility third-party libraries 149
 - use of non-optimized images 149
 - use of unmanaged networking connections 149
- common optimization mistakes**
 - about 199
 - design mistakes 200
 - incorrect usage of game services 200
 - programming mistakes 199, 200
 - wrong game data structure 200
- ComposeShader** 160
- Concurrent Mark Sweep (CMS)** 219
- Concurrent Partial Mark Sweep (CPMS)** 219
- Concurrent Sticky Mark Sweep (CSMS)** 219
- Corona**
 - about 72
 - cons 72
 - pros 72
- cost per action (CPA)** 312
- cost per click (CPC)** 312
- cost per impression (CPI)** 312
- CPU architectures, supported by NDK**
 - ARM 270
 - MIPS 271
 - Neon 271
 - x86 271
- Crashlytics**
 - about 306
 - reference 305
- cross-platform tools**
 - about 67
 - Cocos2d-x 68
 - Corona 72
 - PhoneGap 71, 72
 - Titanium 73
 - Unity3D 69, 70
 - Unreal Engine 70, 71
- custom shaders**
 - writing 161
- Cygwin** 258

D

- Dalvik Debug Monitor Server (DDMS)** 55, 56
- Dalvik message log** 218
- Dalvik Virtual Machine (DVM)** 2
- data file optimization** 177
- debugging for Android**
 - while working with cross-platform engines 230
- design optimization**
 - about 177
 - game design optimization 177
 - technical design optimization 178
- design polishing**
 - about 276
 - game difficulty balance 277
 - game economy balance 276
 - game flow, polishing 276

- metagame, polishing 276
- UX, designing 276
- development insights, on Android consoles** 42
- development insights, on Android mobiles** 35-37
- development insights, on Android tablets** 38, 39
- development insights, on Android TV and STBs**
 - overscan 41
 - UI and game design 41
- development insights, on Android watches**
 - about 42
 - correct libraries, including in project 44
 - hardware compatibility issues,
 - with Android versions 44
 - wearable application, creating 43
 - wearable application, setting up 43
- development polishing**
 - about 274
 - memory optimization 274
 - performance optimization 274
 - portability 275
- device configuration options, Android**
 - about 152
 - battery capacity 153
 - display quality 153
 - GPU 153
 - processor 152
 - RAM 152
- disadvantages, C++ for games**
 - about 262
 - high program complexity 262
 - manual memory management 263
 - platform dependent compiler 263
- Draw 9-Patch** 58
- drawable resources** 192
- DXTC** 267

E

- Eclipse, for Android development**
 - about 56
 - benefits 57
 - drawbacks 57

- effective cost per mile (eCPM)** 312
- example smart TV**
 - specifications 26
- exception handling, in Android games**
 - about 224
 - scope 226
 - syntax 224, 225
- exceptions, in game development process**
 - arithmetic exceptions 228
 - custom exceptions 229
 - index out of bound exceptions 227, 228
 - input/output exceptions 228
 - network exceptions 229
 - null pointer exceptions 226, 227

F

- features and support, Android game development**
 - Android device hardware configuration 3
 - direct manipulation interface 3
 - excellent support, of multimedia 3
 - virtual reality 3
- fields, virtual reality (VR)**
 - architectural design 236
 - education and learning 236
 - fine arts 236
 - medical therapy 237
 - motion pictures 236
 - urban design 236
 - video games 235
- fillrate** 313
- first-person camera** 145
- fixed camera** 145
- Flurry**
 - about 306
 - reference 305
- Forge TV** 25
- FPS system**
 - about 110-112
 - controlling 116
- frame rate** 7
- frames per second (FPS)** 101
- Freebox Mini 4K** 25
- full length ads** 315

G

GameAnalytics

- about 306
- reference 305

game controls

- about 244
- control placement 246
- Fuse Button 244
- fuse button indication 245
- types 244
- visual countdown 245

game design optimization 177

game design standards

- about 85
- artificial intelligence 86
- art style 86
- change log 87
- game elements 86
- game overview 85
- gameplay details 85
- game progression 86
- level design 86
- storyboard 86
- technical reference 87

game development, for VR devices

- about 239
- VR game design 239
- VR game development constraints 240
- VR target audience 239

game loop

- about 7, 94
- frames, rendering 96
- game update 95
- state update 96
- user input 94, 95

gameplay programming 78

GamePop 32

game portability

- about 283
- multiple hardware configurations, supporting 284
- screen sizes, supporting 283

game programming specifications

- about 78
- gameplay programming 78

- graphics programming 79
- technical programming 79

game promotion channels

- about 320
- Android forums 320
- Facebook promotion 321
- sports forums 320
- Twitter, and other social platforms 321
- YouTube channels 320

game revenue, planning

- about 316
- regional variations, of revenue plan 318
- revenue sources 317
- revenue, versus profit 317

game revenue, sources

- about 317
- advertisement revenue 317
- in-app purchase revenue 317
- other sources 318

games

- versus applications 5

game state machine

- general idea 107, 110

Game Stick 31

game tool programming 80

game update 102, 103

geometry shaders 159

Google Analytics

- about 288
- best utilization 289
- integration tips 289
- significance 288

Google Cloud Messaging (GCM) 293

Google Daydream 238

Google IAB

- about 289
- advantages 290
- disadvantages 291
- integrating 290

Google IAB model

- about 289
- consumable items 290
- non-consumable items 290
- subscriptions 290

Google Leaderboard

- about 291
- integrating 291

- significance 291
- variations 292
- Google Nearby** 15
- Google Play Services** 288
- Gradle** 258
- graphics programming** 79

H

- hardware dependency**
 - about 112
 - display 113
 - logical operations 114
 - memory load/unload operations 113
 - rendering 113
- HDPI** 153
- heap memory** 113, 194, 195
- Hierarchy Viewer** 57
- HTC Dream** 17

I

- in-app purchase options**
 - about 308
 - career billing services 309
 - store billing services 308
- in-app purchases**
 - about 307
 - consumable items 310
 - non-consumable items 310
 - subscriptions 310
 - types 310
- industry best practices**
 - about 89
 - design standards 89
 - programming standards 90
- in-game dynamic advertisements** 315
- instrumented tests** 232
- interrupt handling** 106
- interstitial advertisements**
 - about 314
 - integration best practice 314, 315

J

- Java** 258
- Java Native Interface (JNI)** 256

L

- layout resources** 192
- LDPI** 153
- leaderboards**
 - displaying, options 292
 - storing, options 292
- LG G Watch**
 - specifications 34
- LG UPlus Android TV** 25
- libraries for game development, on wearable devices**
 - notifications 44
 - Wearable Data Layer 44
 - Wearable UI support library 44
- LinearGradient** 160
- local network multiplayer** 302
- local test** 231
- Localytics**
 - about 307
 - reference 306
- log messages**
 - about 218
 - ART message log 218
 - Dalvik message log 218
- Lower CamelCase** 90

M

- Mad Catz MOJO** 32
- MDPI** 153
- memory footprint, monitoring**
 - about 217
 - heap update, checking 219, 220
 - log messages, checking 218
 - memory allocation, tracking 220, 221
 - memory leaks, tracking 222
 - overall memory usage, checking 221
- memory load/unload operations**
 - heap memory 113
 - Read-only memory (ROM) 114
 - register memory 114
 - stack memory 114
- memory management, in Android**
 - about 186
 - application memory distribution 188
 - memory allocation and deallocation 187
 - shared application memory 187

- memory optimization**
 - about 178
 - significance 195, 196
 - tricks 179-182
- memory segments**
 - about 193
 - heap memory 194, 195
 - register memory 195
 - stack memory 193
- menu resources 192**
- meta design 303**
- microconsoles 28**
- Micromax Bolt A24**
 - configuration specification 18, 19
- Microprocessor without Interlocked Pipeline Stages (MIPS) 271**
- Mixpanel**
 - about 306
 - reference 306
- mobile game loop, with touch interface**
 - working 103
- mobile phones**
 - market shares, since 2012 21
- modern age Android console**
 - specifications 29
- modern VR systems 235**
- monetization aspects, analytic tools**
 - about 304
 - advertisement display, counting 305
 - advertisement display, tracking 305
 - likes and dislikes, identifying of users 305
 - metagame, improving 305
 - metagame, validating 305
 - paying users, tracking 305
 - popular regions, identifying of game 304
- monetization techniques**
 - about 315
 - freemium model 316
 - free model 316
 - premium model 316
 - try-and-buy model 316
- moving camera 145**
- multiplayer implementation**
 - about 299
 - local network multiplayer 302
 - pass and play turn-based multiplayer 301

- real-time multiplayer 299, 300
 - single-screen real-time multiplayer 301
 - turn-based multiplayer 300, 301
- multiple architecture support**
 - integration, advantages 271
 - integration, disadvantages 272

N

- native code performance 264**
- native project build configuration**
 - about 258
 - Android.mk configuration 258, 259
 - Application.mk configuration 260
- native shared library 256**
- native static library 257**
- NativeTrackTM 306**
- Neon architecture 271**
- network programming 80**

O

- OgleBox Android TV 25**
- OpenGL**
 - texture compression 267
 - used, for rendering 265
- OpenGL 1.x 265**
- OpenGL 2.0 265**
- OpenGL 3.0 266**
- OpenGL 3.1 266**
- OpenGL manifest configuration 268, 269**
- OpenGL rendering system 146**
- OpenGL version**
 - detecting 266
 - setting 266
- OpenGL versions**
 - about 265
 - OpenGL 1.x 265
 - OpenGL 2.0 265
 - OpenGL 3.0 266
 - OpenGL 3.1 266
- optimization fields, in Android games**
 - design optimization 177
 - memory optimization 178
 - performance optimization 183
 - resource optimization 176

Ouya

- about 28
- specifications 28

overall performance optimization

- about 196
- base resolution, selecting 196
- database management 197
- frame rate, increasing 198
- network connection management 198
- portability range, defining 197
- program structure 197

P

pass and play turn-based multiplayer 301

performance, and memory

- balance between 115
- relation between 186

performance optimization

- about 183
- significance 198, 199
- tricks 183-185

performance profiling tools 64, 65

PhoneGap

- about 71
- cons 72
- pros 71

pixel shaders 159

platform-specific specialties

- about 44
- Android consoles 46
- Android mobiles 45
- Android STBs 45
- Android tablets 45
- Android televisions 45
- Android watches 46

play testing

- about 277
- gameplay, adopting 279
- monetization 278, 279
- smoothing running, of game 279
- user actions, during gameplay 278
- user actions, while browsing game 278
- user gameplay difficulty levels 277
- user retention 280

polishing

- art polishing 275

- design polishing 276

- development polishing 274

- requisites 274

private RAM 221

processing segments, in Android

- about 188
- application priority 188
- application services 191
- resource processing 191

ProGuard 59, 182

Proportionate Set Size (PSS) 187

Public Leaderboard 292

push notifications

- application integration 295, 296
- database 293
- GCM service 294
- GCM setup 298
- integrating 295
- server 293
- server setup 298
- target device 293

push notifications, significance

- about 298
- alternative communication channel 299
- user behavior, knowing 299
- user control 299
- user retention 298

PVRTC 267

Python 258

R

RadialGradient 160

Razor Forge TV

- specifications 30

read-only memory (ROM) 114

real-time multiplayer 299, 300

real-time operating systems (RTOS) 252

regional variations, of revenue plan

- about 318
- user base variations 319
- user behavior variations 319

register memory 114, 195

rendering pipeline, in Android

- 2D rendering pipeline 145
- 3D rendering pipeline 146

- requisites, analytic tools**
 - game balancing 303
 - game crash reports 303
 - game event triggers 303
 - gameplay frequency 303
 - gameplay session timing 303
 - piracy prevention 304
 - user behavior 303
 - user retention 304

- research and development programming 81**

- resource optimization**
 - about 176
 - art optimization 176
 - data file optimization 177
 - sound optimization 177

- resource processing**
 - about 191
 - color resources 192
 - drawable resources 192
 - layout resources 192
 - menu resources 192
 - other resources 192
 - tween animation resources 192

- RISC (Reduced Instruction Set Computing) 270**

- rotating camera 145**

S

- sample game loop**
 - creating, Android SDK used 97-101

- Samsung Galaxy S6**
 - configuration specification 19

- services, Google Play Services package**
 - Google Analytics 288
 - Google IAB 289
 - Google Leaderboard 291
 - push notifications 293

- SFXs 281**

- shaders**
 - about 156
 - benefits 157
 - consequences 157
 - geometry shaders 159
 - in 2D game space 169

- in 3D game space 170, 171
 - necessity 156
 - pixel shaders 159
 - scope 158
 - tessellation shaders 159
 - through OpenGL 163-169
 - types 159
 - using, in games 169
 - vertex shaders 159
 - working 158

- shaders, in games**
 - cons 173
 - pros 173

- shared application memory 187**

- Shield Android TV 25**

- short length ads 315**

- single-screen real-time multiplayer 301**

- SmartTags 306**

- Social Leaderboard 292**

- Sony Xperia Z4**
 - configuration specifications 24

- sound effects, UX polishing**

- about 281
 - SFXs 281
 - theme music 281

- sound optimization 177**

- sound programming 80**

- stack memory 114**
 - about 193
 - working 193

- store billing services**
 - about 308
 - Amazon billing services 309

- strategic placement, of debug statements**

- about 222
 - memory allocation 222
 - object state, tracking at runtime 223
 - object values, tracking 223
 - program flow, checking 223

- styles, for different development engines**
 - about 88

- programming languages 88
 - target platforms 89
 - work principles 88

- SweepGradient 160**

T

target device configuration, Android

- feature requirement 9
- game scale 8
- scope for portability 10
- selecting 8
- target audience 9

target OpenGL ES version, selection factors

- about 269
- device support 269
- performance 269
- programming comfort 270
- rendering feature 270
- texture support 269

technical design optimization 178

technical design standards

- about 81
- change log 84
- design pattern 82
- flow diagram 82
- game analysis 82
- other requirements 83
- resource analysis 83
- risk analysis 84
- scope analysis 84
- technical specification 82
- testing requirements 83
- tools 83

technical programming

- about 79
- game tool programming 80
- network programming 80
- research and development
 - programming 81
- sound programming 80

tessellation shaders 159

testing techniques

- about 231
- instrumented test 232
- local test 231

texture compression, OpenGL

- about 267
- ATC 267
- DXTC 267
- PVRTC 267

theme music 281

third-person camera 145

Titanium

- about 73
- cons 73
- pros 73

tools, for testing

- about 61
- test case, creating 61
- test fixture, setting up 61, 62
- test methods, adding to verify
 - activity 63, 64
- test preconditions, adding 63

transaction effects, UX polishing 281

turn-based multiplayer 300, 301

tween animation resources 192

U

UI polishing 275

Unity3D

- about 69
- cons 70
- pros 69

Unreal Engine

- about 70
- cons 71
- pros 70

Upper CamelCase 90

user acquisition techniques

- about 319
- forum discussions 321
- game blogs 321
- game promotion channels 320
- other techniques 322
- paid user acquisition 321

user retention techniques

- about 322, 323
- daily bonus 323
- frequent updates 324
- leaderboards and achievements 323
- offerwall Integration 323
- push notifications 323

UX polishing

- about 280
- action feedback 281
- sound effect 281

transaction effects 281
visual effects 280

V

variations, Google Leaderboard

Public Leaderboard 292
Social Leaderboard 292

vertex shaders 159

video advertisements 315

full length ads 315
short length ads 315

virtual reality therapy (VRT) 237

virtual reality (VR)

about 234
evolution 234
fields 235-237

visual effects, UX polishing 280

VR development 252

VR game design 239

VR game development constraints 240

VR game development, through Google VR

about 246
Android NDK used 248
Android SDK used 246

VR game, exiting

Back button, hitting 242
Home button, hitting 243

VR gaming concepts 252

VR target audience 239

X

x86 architecture 271

XHDPI 153

XXHDPI 153

XXXHDPI 153

Z

zipalign 182

